UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETROTÉCNICA
ENGENHARIA DE CONTROLE E AUTOMAÇÃO

LUCAS VILELA SANCHES DE MAMANN
NÍKOLAS NASCIMENTO AGUILAR
RAUL MATHEUS MARTINS

AUTOMAÇÃO DE DISPOSITIVOS PARA CAPTURA DE JAVALIS COM BASE EM VISÃO COMPUTACIONAL

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA

2017

LUCAS VILELA SANCHES DE MAMANN
NÍKOLAS NASCIMENTO AGUILAR
RAUL MATHEUS MARTINS

AUTOMAÇÃO DE DISPOSITIVOS PARA CAPTURA DE JAVALIS COM BASE EM VISÃO COMPUTACIONAL

Trabalho de Conclusão de Curso de Graduação apresentado à disciplina de Trabalho De Conclusão De Curso 2, do Curso Superior de Engenharia de Controle e Automação do Departamento Acadêmico de Eletrotécnica - DAELT - da Universidade Tecnológica Federal do Paraná - UTFPR, como requisito parcial para obtenção do título de Engenheiro de Controle e Automação.

Orientador: Prof. Dr. Guilherme Luiz Moritz

CURITIBA

2017

Lucas Vilela Sanches De Mamann
Nikolas Nascimento Aguilar
Raul Matheus Martins

Automação de dispositivos para captura de javalis com base em visão computacional

Este Trabalho de Conclusão de Curso de Graduação foi julgado e aprovado como requisito parcial para a obtenção do Título de Engenheiro de Controle e Automação, do curso de Engenharia de Controle e Automação do Departamento Acadêmico de Eletrotécnica (DAELT) da Universidade Tecnológica Federal do Paraná (UTFPR).

Curitiba, 5 de dezembro de 2017.

_____
Prof.Paulo Sérgio Walenia , Eng.
Coordenador de Curso de
Engenharia de Controle e Automação

_____
Prof. Marcelo de Oliveira Rosa, Dr.
Responsável pelos Trabalhos de Conclusão de Curso
de Engenharia de Controle e Automação
do DAELT

ORIENTAÇÃO

_____
Guilherme Luiz Moritz, Dr.
Universidade Tecnológica Federal do Paraná
Orientador

BANCA EXAMINADORA

_____
Guilherme Luiz Moritz, Dr.
Universidade Tecnológica Federal do Paraná

_____
Elder Oroski, Dr.
Universidade Tecnológica Federal do Paraná

_____
Marcelo de Oliveira Rosa, Dr.
Universidade Tecnológica Federal do Paraná

A folha de aprovação assinada encontra-se na Coordenação do Curso de Engenharia de Controle e Automação.

RESUMO

MAMANN, Lucas V. S. de; AGUILAR, Níkolas N.; MARTINS, Raul M. AUTOMAÇÃO DE DISPOSITIVOS PARA CAPTURA DE JAVALIS COM BASE EM VISÃO COMPUTACIONAL. 158. Trabalho de Conclusão de Curso – Engenharia de Controle e Automação, Universidade Tecnológica Federal do Paraná. Curitiba, 2017.

Este projeto tem como objetivo a criação de uma armadilha, que seja acionada automaticamente por meio de um sistema de controle baseado em técnicas de visão computacional. Ela tem capacidade de fazer o reconhecimento através de imagens de seus alvos, no caso deste trabalho javalis, aguardando pelo momento mais oportuno para que a maioria dos indivíduos identificados, se não todos, possam ser capturados. O sistema também conta com uma interface com o usuário, para que este possa operar a armadilha e, portanto necessita de um meio de transmissão de dados de longa distância, cujo desenvolvimento está fora do escopo do projeto. Os *softwares* elaborados neste trabalho têm como base algoritmos de processamento de imagens.

Palavras-chave: Armadilha, Javali, Visão Computacional, Processamento de Imagens

ABSTRACT

MAMANN, Lucas V. S. de; AGUILAR, Níkolas N.; MARTINS, Raul M. AUTOMATION OF WILD BOAR CAPTURE DEVICES BASED ON COMPUTER VISION. 158. Trabalho de Conclusão de Curso – Engenharia de Controle e Automação, Universidade Tecnológica Federal do Paraná. Curitiba, 2017.

The objective of this project is to build a trap, which is automatically released by a control system based on computer vision techniques. It is capable of recognising on images its targets, being wild boars for this study, waiting for an opportune moment so that most of the identified individuals, or all of them, can be captured. The system also has an user interface, allowing them to operate the trap and thus need a long range data transmission method, which development is outside of this project scope. The softwares elaborated on this work have image processing algorithms as its base.

Keywords: Trap, Wild Boar, Computer Vision, Image Processing

LISTA DE FIGURAS

LISTA DE SIGLAS

ANN	*Artificial Neural Network*
Caffe	*Convolutional Architecture for Fast Feature Embedding*
DNN	*Deep Neural Network*
GPIO	*General-Purpose Input/Output*
GPRS	*General Packet Radio Service*
HOG	*Histogram of Oriented Gradients*
I2C	*Inter-Integrated Circuit*
IBAMA	Instituto Brasileiro do Meio Ambiente e dos Recursos Naturais Renováveis
LBP	*Local Binary Patterns*
OpenCV	*Open Source Computer Vision Library*
PCL	*Point Cloud Library*
RAM	*Random-Access Memory*
RGB	*Red - Green - Blue*
SIFT	*Scale-Invariant Feature Transform*
SoC	*System on Chip*
SPI	*Serial Peripheral Interface*
UART	*Universal Asynchronous Receiver/Transmitter*
USB	*Universal Serial Bus*
API	*Application Programming Interface*
SSH	*Secure Shell*

SUMÁRIO

1 INTRODUÇÃO

De acordo com a Convenção sobre Diversidade Biológica, espécies exóticas são quaisquer tipo de seres vivos introduzidos, geralmente por humanos, em um *habitat* diferente do seu original. Estas espécies, porém, só se tornam invasoras quando conseguem superar os organismos nativos na luta pela sobrevivência e se espalhar a ponto ameaçar o ecossistema, ou seja, quando sobrevivem e prosperam em seu novo *habitat* (CONVENTION ON BIOLOGICAL DIVERSITY, 2011).

No Brasil, assim como em diversos lugares do mundo, uma destas espécies exóticas invasoras é o javali *Sus scrofa*, nativo da Europa, Ásia e do norte da África. Acredita-se que sua disseminação no Brasil se iniciou por volta de 1989 pelo Rio Grande do Sul, quando populações de javali provindas de cativeiros uruguaios, de onde foram soltas ou escaparam, cruzaram a fronteira com o Brasil (KAIZER et al., 2014). Outra hipótese é de que tenham entrado pelo Pantanal a mais de duzentos anos (PEDROSA et al., 2015a).

Os javalis foram capazes de se espalhar pelo Brasil, devido à variedade de sua dieta, à sua grande capacidade de adaptação (KAIZER et al., 2014) e alta reprodutividade (WALLAU et al., 2016), tendo sido encontrados em todos os estados das regiões sul, centro-oeste e sudeste do Brasil, além do sul da Bahia (PEDROSA et al., 2015a). O mapa mostrando tal distribuição pode ser visto na Figura 1, em que em amarelo estão as regiões pantaneiras por onde os javalis adentraram o país, em vermelho a região de fonteira com o Uruguai por onde houve a invasão em 1989 e em verde são os locais onde foram detectados javalis em 2007.

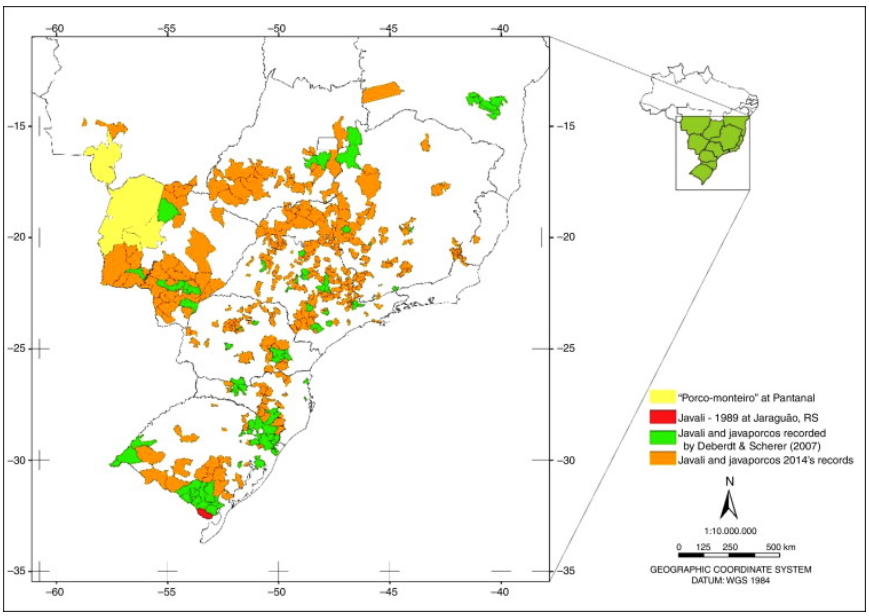

Figura 1: Distribuição atual de javalis selvagens no Brasil.
Fonte: PEDROSA et al. (2015b).

Não é somente o fato de os javalis selvagens terem se alastrado rapidamente pelo país que preocupa as autoridades, são também os problemas econômicos e de saúde que eles trazem consigo. Segundo Korn e Bomford (1996), os javalis causam muitos danos à agropecuária e à natureza, pois destroem lavouras e vegetação nativa, utilizando-as como seu alimento, pisoteando plantas e cavando o solo em busca de raízes; além disso, predam animais nativos da região e animais de rebanho; e também competem com outros animais por alimentos. Ademais, os javalis podem ainda ser vetores de parasitas e doenças exóticas ou endêmicas, como leptospirose, brucelose e tuberculose, contaminando tanto os animais quanto os humanos.

Devido aos problemas citados que são causados pelos javalis e em razão de sua proliferação pelo território nacional, o IBAMA (Instituto Brasileiro do Meio Ambiente e dos Recursos Naturais Renováveis) aprovou a instrução normativa nº 03/2013, permitindo o controle populacional de javalis vivendo em liberdade em todo o Brasil (IBAMA, 2013).

A caça e outros meios que retirem da natureza apenas uma pequena porcentagem dos javalis, não são muito eficientes na diminuição da população destes animais, como bem representa Wallau et al. (2016). Uma simulação utilizando dados de crescimento conservadores mostra que, em um período de 4 anos, uma população inicial de 10 javalis pode dar origem a 500 animais. Já outros estudos realizados no Rio Grande do Sul, mostram que no mesmo período, 10 javalis podem procriar a ponto de chegar a uma população de quase 2000 animais (WALLAU et al., 2016). As simulações de Wallau et al.

(2016) também indicam que se for realizado um controle de somente 30% da população, em 4 anos a população inicial de 10 animais cresceria para 600, porém em mais 6 anos já alcançaria 10.000 indivíduos. Por outro lado, ele demonstra que para um controle de 70% da população, ela teria um crescimento mínimo em 4 anos, aumentando de 10 para 20 animais.

Percebe-se portanto, tendo como base os estudos de Wallau et al. (2016), que a caça manual de javalis dificilmente terá algum efeito de redução da população, visto que a população javalis vivendo livres atualmente no Brasil já é demasiadamente grande.

1.1 TEMA

Construção de armadilha não letal para capturar espécies invasoras em ambientes agrícolas, que funcione de maneira autônoma.

Apesar de já haver no mercado armadilhas para a captura de javalis, como a americana *BoarBuster* mostrada na Figura 2, elas não se enquadram à realidade brasileira. Os dispositivos existentes necessitam de conexão direta com a internet, para que o usuário faça o acionamento remoto da armadilha via celular. Este projeto visa, portanto, remover a necessidade de que um usuário faça o desarme remoto e também permitir que a armadilha seja instalada em locais sem acesso à internet, como é o caso da maioria das propriedades rurais brasileiras.

Figura 2: Armadilha *BoarBuster* acionada remotamente via internet.
Fonte: BOARBUSTER (2016).

1.2 DELIMITAÇÃO DO TEMA

Desenvolvimento de sistema de automação para armadilha não letal capaz de identificar grupo de javalis, utilizando tecnologia de visão computacional e tecnologias complementares; ativar a armadilha quando uma porcentagem mínima, configurável, da área da armadilha estiver ocupada por javalis, através do envio de comando de ativação de solenóides; oferecer capacidade para informar status ou acionamento via comunicação por rádio e/ou GPRS.

1.3 PROBLEMA E PREMISSAS

Não é escopo deste trabalho o projeto ou especificação de estruturas mecânicas da armadilha, que para o desenvolvimento será abstraída como uma estrutura suspensa capaz de permanecer elevada durante tempo indeterminado e que possa ser "solta"ao ser comandada, assim como a estrutura apresentada na Figura 2.

A dificuldade a ser resolvida concentra-se no tratamento e processamento de imagens para identificação da espécie alvo, neste caso javalis. Bem como na geração de um controle da armadilha em função da identificação.

1.4 OBJETIVOS

1.4.1 OBJETIVO GERAL

Projetar uma armadilha com acionamento autônomo, que captura animais presentes em uma área, evitando causar a estes ferimentos. A armardilha utiliza recursos de visão computacional e sensoriamento para definir se há movimentação dentro de sua área de atuação e, utiliza-se da informação de porcentagem de ocupação aproximada de sua área para realizar seu acionamento.

Não cabe a este trabalho determinar os parâmetros de controle tais como número exato de individuos ou distância dos mesmo do aparato. Tais configurações devem levar em conta estudos comportamentais dos javalis para determinar, por exemplo, seu tempo de reação ao acionamento da armadilha, capacidades de coleta e processamento de dados muito elevada, o que foge do escopo deste projeto. O sistema, no entanto, oferece a possiblidade de configurar certos parâmetros que influenciam no número mínimo de indivíduos a serem capturados.

1.4.2 OBJETIVOS ESPECÍFICOS

- Especificação de uma câmera USB, interface 2.0, ou IP, conectada a um sistema embarcado, para coletar imagens da área da armadilha e seu entorno;
- Implementação de software que identifica movimentação dentro da armadilha, com base em visão computacional e sensoriamento;
- Extração de dados pertinentes sobre o status da armadilha, a partir de imagens e sensores;
- Desenvolvimento do controle de acionamento automático da armadilha.

1.5 JUSTIFICATIVA

A falta de predadores e seu papel como hospedeiro de várias doenças (KORN; BOMFORD, 1996), configura os Javalis como um vetor de problemas de saúde em potencial para a população humana. Esta espécie invasora também mostra-se como um grande risco para os ambientes onde foi inserida, pois ela costuma ser extremamente agressiva, podendo destruir a vegetação nativa, predar animais silvestres e até mesmo ameaçar seres humanos que venham a se deparar com eles.

Outro ponto relevante é o fato de que o setor agropecuário tem elevada importância para a economia do Brasil, e os javalis estão causando muitos prejuízos a este ramo da economia. A destruição de plantações e predação de rebanhos afeta tanto o mercado interno quanto o externo, pois modifica o preço das *commodities* na bolsa de valores e sua prospecção econômica podendo causar uma inflação no preço da comida.

A valoração da parte técnica desse projeto se dá justamente na análise da visão computacional, pois é preciso reconhecer o objeto a ser capturado através de uma biblioteca gráfica que comparará os dados recebidos com uma base de dados, ou seja, a teoria de reconhecimento de padrões terá que ser levada em consideração para montagem desse sistema de captura.

Entende-se ainda que o algoritimo seja tão flexivel quanto possivel, de maneira que seja possível inclusive expandí-lo para o reconhecimento de outros animais, dado que o sistema seja previamente treinado de acordo, oferecendo uma solução mais global, e não somente pontual a uma determinada região.

O conhecimento de que a população corre o risco de ser atingida por doenças, de

que o mercado de ações é dependente da venda de produtos primários e que o combate à esses animais por meio de caça não é eficiente o suficiente para conter o avanço de sua população, são grandes motivações para o desenvolvimento do sistema a que este trabalho se propõe.

Ao considerar os conhecimentos técnicos obtidos durante o curso de Engenharia de Controle e Automação e os aplicados neste trabalho é possivel citar: Acionamento de Sistemas Elétricos, Sinas e Sistemas, Eletrônica, Sistemas Microcontrolados, Redes Industriais e Supervisão De Processos.

1.6 PROCEDIMENTOS METODOLÓGICOS

Para o desenvolvimento do projeto proposto, foi necessário realizar pesquisas principalmente sobre teoria de visão computacional. As pesquisas no campo de visão computacional foram mais voltadas à decisão de quais método de identificação de objetos e detecção de movimento seriam os mais eficientes. Também foi preciso recorrer às documentações disponíveis a respeito da biblioteca de processamento de imagem utilizadas.

Após o desenvolvimento dos softwares necessários, foi testada sua eficácia. Para realizar testes iniciais, foi utilizado um ambiente de mais fácil controle, utilizando objetos de menor porte.

Com as etapas anteriores completadas, ainda espera-se coletar diversas imagens dos alvos do software de reconhecimento, neste caso de javalis. Independente da técnica escolhida para classificar e detectar as imagens, será usado vídeo para extrair o maior número possível de *frames*, gerando um banco de imagens suficientemente grande para treinar o algoritimo escolhido e aumentar sua eficiência.

2 ESCOPO DO TRABALHO

2.1 TRABALHOS RELACIONADOS

Há alguns trabalhos já publicados sobre detecção de animais baseada em visão computacional, com diversos enfoques e objetivos. (SHARMA; SHAH, 2016) por exemplo, trabalham com a detecção de vacas em específico em ambientes urbanos e com cálculo de distância, por pixels, do animal detectado, utilizando Histogramas de Gradientes Orientados (HOG) e Cascata de Classificadores, *Cascade Classifiers*. (LÉVESQUE; BERGEVIN, 2010), por sua vez, utilizam imagens estéreo e características como cor, textura, e modelo 3D para identificar animais na natureza, utilizando-se do C++ e OpenCV para aplicar técnicas de segmentação de fundo, extração de características e triangulação de pontos 3D. Ainda a respeito da identificação de animais na natureza, (YU et al., 2013) fazem a identificação de diferentes espécies de animais fotografados por câmeras disparadas por movimento, utilizando também extração de características com algoritmos HOG, LBP e SIFT.

Apesar de também se tratar de um projeto que envolve a detecção de animais em seu *habitat*, na natureza, o que este trabalho propõe de novidade em relação aos anteriores é utilização da visão computacional para gerar dados para um sistema de controle e não apenas para a pura identificação do animal. O objetivo é identificar movimentação nas imagens dentro da armadilha, para desarmá-la quando isto ocorrer. Adicionalmente, pode-se identificar que a movimentação é causada por javalis. Pretende-se utilizar técnicas de detecção de movimento em imagens, para desarmar a armadilha e, possivelmente, utilizar a teoria de redes neurais o reconhecimento de imagens, diferentemente dos trabalhos citados.

Os métodos propostos serão brevemente abordados na seção 3.

2.2 MATERIAIS E MÉTODOS

Um dos maiores desafios relacionados a este trabalho é o de identificar os objetos desejados em uma imagem, para posteriormente passar por uma etapa de controle e tomada de decisão quanto à ativação da armadilha. Portanto, esta será a etapa de maior enfoque deste trabalho.

Nas seções 2.2.1 e 2.2.2, serão descritas opções existentes de *software* e *hardware* para o desenvolvimento do projeto, tanto para o processamento de imagem quanto para o controle da armadilha. Na seção 2.2.3, serão discutidos os métodos de obtenção de dados de entrada para o sistema, a fim de se realizar o devido processamento sobre eles. Também serão expostas as motivações para as escolhas que serão feitas em relação a estes três itens. E, por fim, na seção 2.2.4 serão apresentadas as formas de saída de dados do sistema.

2.2.1 DO SOFTWARE

Há algumas linguagens de programação e bibliotecas que implementam processamento de imagem e detecção de objetos, bem como opções de softwares prontos. Um *software* bastante difundido para a solução de problemas de engenharia em geral é o MATLAB, que dispõe também de pacotes para aplicações de processamento de imagens (MATHWORKS, 2017). Existem ainda, alguns *frameworks* específicos para o reconhecimento de imagens via redes neurais, como o Caffe (JIA et al., 2014) e o TensorFlow (TENSORFLOW, 2017).

Quanto às linguagens de programação, as mais utilizadas para este fim estão C, C++ e Python. Para elas, existem bibliotecas que disponibilizam funções para manipulações gerais de imagens, tais como OpenCV e PCL. É possível ainda encontrar outras bibliotecas, que realizam funções específicas como identificação de objetos, como a *Open Detection*, porém tais bibliotecas geralmente utilizam a OpenCV como base para seu código (OPEN DETECTION, 2017).

Dentre as opções apresentadas, o MATLAB é a solução mais fácil de ser trabalhada, visto que a linguagem de programação é de mais alto nível e a quantidade de código para fazer reconhecimento de objetos é menor. Apesar de ser uma opção vantajosa em termos de trabalho necessário, ela não é apropriada para esta aplicação, pois o MATLAB necessita de grande capacidade de processamento, algo que estará indisponível para a plataforma que se deseja utilizar, uma vez que o sistema em desenvolvimento será

embarcado e estará em áreas remotas. Sem mencionar que este *software* é proprietário e os custos para adiquirí-lo encareceriam demasiadamente o projeto.

Como já foi dito, as opções de bibliotecas exigem mais trabalho para o seu desenvolvimento, porém têm a vantagem de terem código aberto e disporem de maior flexibilidade, podendo-se fazer uma programação específica para este trabalho, de acordo com o que for necessário. Apesar de ambas bibliotecas mencionadas, OpenCV e PCL, serem utilizadas para processamento de imagens e disporem de técnicas avançadas para tal, decidiu-se pelo uso da OpenCV, devido à vasta documentação (OPENCV, 2017d), além de ser nativamente suportada por algumas plataformas embarcadas, discutido na seção 2.2.2.

Por fim, a linguagem de programação escolhida para o desenvolvimento do *software* deste projeto foi o C++. Em comparação ao C, o C++ possui mais recursos, com maior complexidade, permitindo realizar mais facilmente tarefas como processamento de imagem, visto que ele é orientada a objetos. Quanto sua comparação à linguagem python, pode-se dizer que ambas estão equiparadas em questão de vantagens para esta aplicação, no entanto, pelo fato de os integrantes deste projeto estarem mais familiarizados à linguagem C++, esta foi a escolhida entre as duas.

Os códigos, que são escritos com essa linguagem, são inseridos no sistema embarcado, que faz a interface com o mundo externo, por meio de sensores e interfaces de comunicação. Dentre as funcionalidades do *software* desenvolvido estão, receber dados do *software* de reconhecimento de imagem, estimar o número de indivíduos dentro da armadilha e fazer a interação com o mundo físico, ou seja, desarmar a armadilha quando um certa condição for atingida.

2.2.1.1 OPENCV

O OpenCV (*Open Source Computer Vision Library*) é uma biblioteca com código aberto de visão computacional e *machine learning*. Ela possui interface com C, C++, Python, Java e MATLAB e pode ser rodada em sistemas Windows, MAC OS, Linux e seus derivados. Esta biblioteca tem a função de facilitar a aplicação de visão computacional e *machine learning* em programas, implementando tanto algoritmos clássicos quanto outros recentes (OPENCV, 2017e).

A biblioteca OpenCV é utilizada não somente para projetos amadores, mas também para projetos profissionais, de grandes companhias mundiais como Google, Mi-

crosoft, Sony, Toyota, entre outras. Suas aplicações vão desde detecção de intrusos em vídeos de vigilância, até monitoramento de equipamentos em minas (OPENCV, 2017e). O OpenCV também pode ser utilizado para reconhecimento facial, rastreamento de movimento, detecção de objetos, além de outros usos.

Neste trabalho, o OpenCV é utilizado para fazer um tratamento inicial das imagens. Com ele são identificadas e extraídas as partes da imagem que sejam candidatas a serem o objeto procurado, javali, através de técnicas de detecção de movimento e rastreamento, que serão mais detalhadas na seção 3.

2.2.2 DA PLATAFORMA

Escolhidas as técnicas para a detecção assim como as bibliotecas usadas para tal se faz necessário definir qual plataforma utilizar.

Uma das premissas desse projeto é oferecer um sistema autônomo que possa ser instalado em áreas afastadas, como plantações. Tais cenários normalmente oferecem baixa conectividade, não sendo possível depender de conexão a um servidor remoto para processamento dos dados. Sendo assim, é preciso recorre a um sistema embarcado capaz de atender esses requisitos.

2.2.2.1 SISTEMA EMBARCADO

A popularização da chamada Internet das Coisas trouxe ao mercado diversos sistemas embarcados para desenvolvimento tais como Arduino, TivaC, Raspberry, BeagleBone entre outros, cada um oferecendo diferentes vantagens em termos de processamento, memória e armazenamento.

Aplicações envolvendo processamento de imagem exigem capacidade de processamento que não pode ser suprida por microcontroladores de baixo consumo tais como PIC e ATMega, utilizados no Arduino, entretanto é importante manter o equilíbrio entre processamento, memória e potência (CARRO; WAGNER, 2003).

Ambas as bibliotecas propostas para esse projeto fazem uso de recursos computacionais disponíveis apenas em sistemas equipados com sistema operacionais como Linux ou Windows, inviabilizando o uso de plataformas microcontroladas tradicionais como o Arduino e TivaC.

Para este trabalho foi escolhido o Raspberry PI 3, motivado pelo suporte ao

sistema Linux, baixo custo, tamanho reduzido, e seu amplo uso com OpenCV.

2.2.3 DA OBTENÇÃO DE DADOS

O sistema recebe dados e os processa em tempo real. A principal fonte de dados serão as câmeras, pois o projeto é mais voltado à visão computacional, ou seja, processamento de imagens para a realização do controle da armadilha. Porém, isto não exclui outras formas de obtenção de dados, como sensores. Os dados entram no sistema através das entradas de dados disponíveis na plataforma de hardware, seja via porta ethernet, USB ou GPIO, dependendo de qual delas o elemento sensor utiliza como sua interface.

2.2.3.1 CÂMERAS

Foram testadas algumas configurações de posicionamento de câmeras para encontrar uma que produz melhores resultados. Isto é, a configuração que propicia o melhor campo de visão da armadilha e seus entorno e que resulta no maior número possível de identificação correta de alvos.

O objetivo é que a(s) câmera(s) capture(m) imagens de no mínimo toda a área interna da armadilha e, se viável, uma certa área ao entorno da mesma, para identificar se não há mais indivíduos prestes a entrarem. Algumas das hipóteses formuladas para serem testadas foram as seguintes:

- Uma única câmera no topo da armadilha, voltada para baixo. Para esta configuração se tornar aceitável, a câmera precisa ser posicionada em um ponto alto o suficiente, de modo que a abertura da imagem atinja toda a região de interesse. Uma vantagem deste modo é o fato de ser preciso trabalhar apenas com um conjunto de imagens e uma perspectiva simples.

- Duas câmeras inclinadas, cada uma cobrindo uma parte da armadilha. Esta configuração traz a desvantagem de ser preciso trabalhar com mais imagens e em perspectiva, além de poder apresentar problemas de superposição de imagens. Em contrapartida, ela oferece um incremento no campo de visão e permite que as câmeras sejam instaladas em posições mais baixas e ainda assim tenham um campo de visão amplo.

Certamente há outros posicionamentos possíveis e utilizando mais câmeras. Porém a complexidade do problema começa a aumentar demasiadamente, pois, quanto maior for

a quantidade de dados entrando no sistema, é preciso mais capacidade de processamento.

Considerando-se os pontos positivos e negativos das configurações propostas, optou-se por utilizar apenas uma câmera no centro da armadilha, voltada para baixo. A sua estruturação física está fora do escopo do projeto, visto que o objetivo é o desenvolvimento do *software*. No entanto, de acordo com as necessidades do trabalho, considera-se que o elemento de fixação será um poste, com sua base fora da área da armadilha e um braço que se estenda até o centro da mesma e, em sua extremidade, estará a câmera. A altura do poste depende da área da armadilha e do ângulo de visão da câmera, ou seja, também não entra no escopo do projeto tal cálculo.

As características da câmera, no entanto, podem ser as mesmas independentemente de qual for a configuração escolhida. A câmera será uma com comunicação via USB ou via ethernet, câmera IP, pois estes são os tipos mais fáceis de serem encontrados para comercialização e que facilitam a obtenção de imagens por parte da plataforma de hardware. Quanto maior seu ângulo de visão melhor, pois maior será a área coberta por uma única câmera. A resolução da câmera impacta diretamente a qualidade das imagens obtidas, porém altas resoluções implicam em maior quantidade de dados a serem tratados e uma menor velocidade de processamento. Recursos adicionais, como detecção de movimento, também são de grande interesse. Entretanto, o essencial é que as características de comunicação sejam atendidas, as outras serão atendidas se a relação entre custo econômico e computacional e benefício estiver dentro do alcance do projeto.

Para as necessidades do projeto, com relação às características da câmera, optou-se por utilizar uma câmera como a *Webcam* HD C270 da Logitech para testes iniciais, verificando-se posteriormente se os resultados obtidos são satisfatórios. Suas especificações técnicas são, 1280 por 960 píxeis de resolução, 1,2 Megapíxeis, interface *High Speed* USB 2.0, campo de visão de 60° e distância focal de 4 milímetros (LOGITECH, 2014).

Todavia, a câmera da Logitech proposta anteriormente, não é adequada para ambientes externos, que será o ambiente de instalação do sistema. Para a aplicação real, seria necessária uma câmera como por exemplo a 3MP IR Bullet Network Camera (DS-2CD2032-I) da Hikvision, que é apropriada para ambientes externos. Suas especificações são superiores às da Logitech, tendo resolução máxima de 2048 por 1536 píxeis, 3 Megapíxeis, interface 100M ethernet, campo de visão de até 79° e distância focal de 4 milímetros (HIKVISION, 2017).

O sistema, entretanto, pode comportar de forma transparente qualquer câmera, pois o foco de todo o trabalho é a parte de software, que não está atrelada às especi-

ficações da câmera. Portanto, o sistema é escalável para equipamentos melhores, sendo independente da escolha de câmera.

2.2.3.2 SENSORES

A utilização de sensores em adição à câmera, pode reduzir o volume de do processamento, melhorando o desempenho do sistema. Isto se deve ao fato destes elementos serem módulos especializados em uma única tarefa e cuja saída exige pouco processamento, porém agrega mais dados para a tomada de decisão pelo controlador.

Um dos tipos de sensores que pode ser de maior interesse para este projeto em específico, é o sensor de presença. Ele pode ser integrado no sistema de forma que, somente sejam processadas imagens quando houver detecção de movimento, reduzindo o processamento para momentos em que não há animais na região da armadilha e, consequentemente reduzindo o consumo de energia.

Neste trabalho não foram utilizados quaisquer tipos de sensores, principalmente devido à natureza dos testes realizados. Tais testes foram feitos com objetos inanimados ou a partir de vídeos, conforme apresentado no capítulo 4, tornando inviável a adição dos sensores de presença.

2.2.4 DA SAÍDA DE DADOS

O sistema, formado pelo *hardware* descrito na seção 2.2.2 e pelo *software* apresentado na seção 2.2.1, se utilizará de dados obtidos da maneira que foi explicada na seção 2.2.3, para aplicar uma estratégia de controle e gerar um sinal digital de acionamento. Por ser um sistema digital, essa saída é apenas um valor alto ou baixo, que pode ser usado conforme for necessário no qual precisa de uma interface física para modificar o sistema físico.

O sistema a ser modificado é a armadilha, que precisará ser acionada no momento em que o controlador julgar correto. Esse momento trata-se do acionamento de motores ou da liberação de travas mecânicas, dependendo de qual for o projeto construtivo da armadilha, que permitirá que a armadilha desça e atinja o objetivo do sistema, aprisionar os javalis. Para tal, o sistema de controle irá se utilizar de uma porta GPIO da plataforma de hardware. Esta porta é justamente a interface necessária, que transforma um sinal digital em um sinal de tensão. O sinal de tensão gerado, por sua vez, será utilizado para acionar um relé ou um contator, dependendo da potência necessária para ligar o elemento

acionador da armadilha.

A outra forma saída de dados que o sistema possui é puramente digital. Ela é a comunicação com o usuário, informando que a armadilha foi ativada. Esta saída é uma curta sequência de dados, que contém apenas um valor digital indicando o ativamento e outro indicando o número de javalis dentro da armadilha. Os dados são enviados por um meio de comunicação sem fio e são traduzidos para uma linguagem inteligível para o usuário através de uma interface compatível, seja web ou um aplicativo de computador ou celular, ambos fora do escopo deste projeto.

2.2.5 CONCLUSÕES SOBRE O SISTEMA

Tendo em vista o que foi exposto neste capítulo, a visão geral da solução proposta é a descrita na Figura 3.

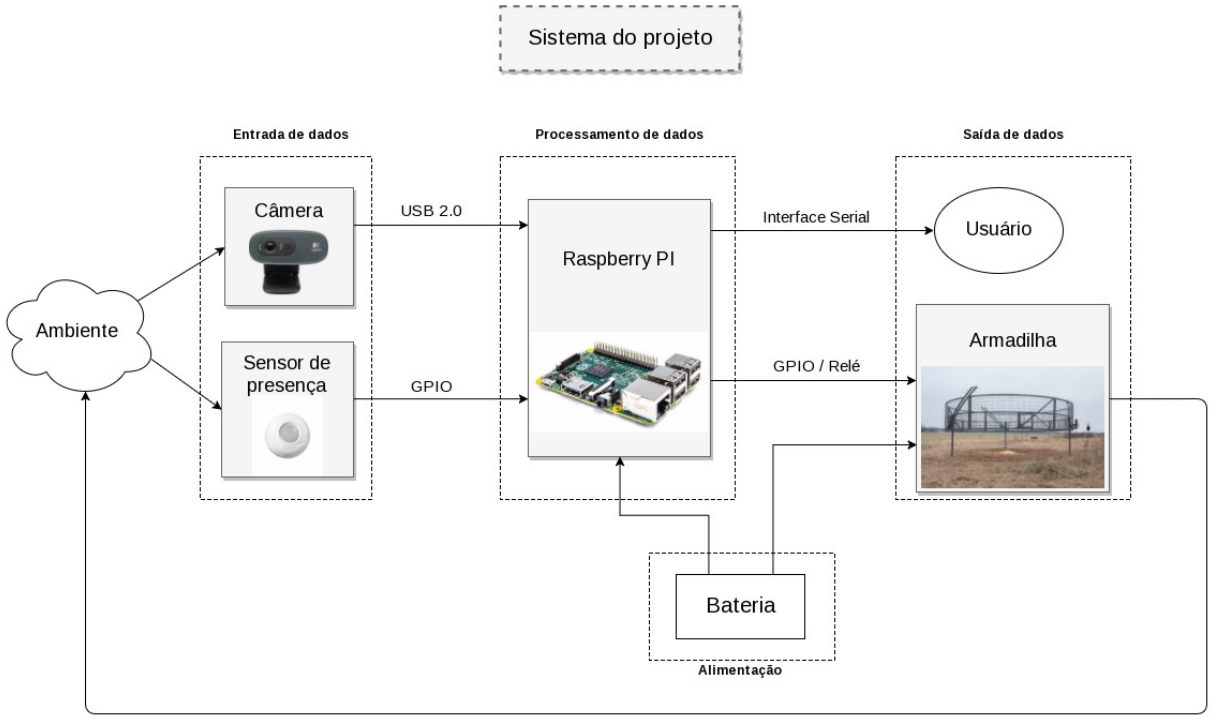

Figura 3: Diagrama de blocos do projeto.
Fonte: Autoria própria.

Conforme apresentado no diagrama de blocos da Figura 3, o sistema é composto basicamente por três partes, de modo que ele obtém dados do ambiente e o modifica de alguma forma. O Ambiente representado na Figura 3, refere-se ao local em que serão retirados os dados, ou seja, do local onde será instalada a armadilha e so qual estarão os animais a serem capturados. O bloco de entrada de dados é a parte responsável por

capturar dados do ambiente e inserí-los no sistema, sendo composto por dois outros blocos, uma câmera e um sensor de presença. O bloco principal, o de processamento de dados, é aquele responsável por receber os dados de entrada e, por meio do processamento das imagens, dados e do sistema de controle, gerar uma saída desejável. Por fim, o bloco de saída de dados é aquele que recebe o sinal de controle e que pode modificar o ambiente, também estando subdividido em dois blocos, o de usuário, que representa a informação que o usuário irá receber a respeito do ativamento da armadilha, e o da armadilha, que irá ser ativada quando necessário.

Entre a entrada de dados e o ambiente, o interfaceamento ocorre por meio dos elementos sensores, que transformam a informação analógica do ambiente em digital. Entre a entrada e o processamento de dados, a interface se dá por meio de um cabo USB, para a câmera, e um fio de dados, para o sensor de presença. Por sua vez, a interface entre o processamento e a saída de dados ocorre de duas formas, via comunicação sem fio para enviar dados ao usuário, e via um par de fios para acionar um relé que comandará a armadilha. E para fechar a malha do sistema, a armadilha faz fisicamente seu interfaceamento com o ambiente, causando uma barreira física quando desarmada, de modo que os animais alvo fiquem restritos a uma certa área.

3 REFERENCIAL TEÓRICO

São necessários alguns conhecimentos na área de processamento de imagens, para a parte de identificação de movimentação nas imagens; e redes neurais, para a parte de reconhecimento de objetos nas imagens. Os conhecimentos sobre processamento de imagens serão descritos a seguir, na seção 3.1.

3.1 TRATAMENTO DE IMAGENS

Após a captura das imagens, para detectar movimentos e a área onde ocorre o movimento via *software*, é preciso tratar as imagens de uma forma específica, que ocorre pela comparação de dois ou mais quadros, *Frames*. Para a realização da comparação é necessário a utilização de algumas técnicas como: subtração de fundo, *Background subtraction*, aplicação de uma faixa limite para análise, *Thresholding*, dilatação, *Dilation*, e erosão da imagem, *Erosion*.

Quanto maior for a dimensão de cada pixel, mais difícil é de fazer o processamento, pois é preciso lidar com mais informações sobre cada *pixel*. Por este motivo geralmente se utiliza imagens em escalas de cinza, reduzindo os dados para apenas uma dimensão por *pixel*, técnica esta que será adotada neste trabalho.

3.1.1 *BACKGROUND SUBTRACTION*

Essa função é um dos passos de processamento mais importantes na área de visão computacional (OPENCV, 2017e), pois ela ajuda a diminuir as informações que as imagens carregam, causando assim uma queda no processamento, fazendo com que o *software* fique mais rápido e com menos erros devido a filtragem feita pela função. Existem várias formas de se fazer a subtração do fundo, mas os parâmetros de entrada são apenas dois: fundo estático, quando o modelo da imagem de fundo não muda no decorrer do tempo, e fundo dinâmico, quando o modelo da imagem de fundo é modificado com o tempo, assim como os objetos móveis na cena.

Simplificadamente, utiliza-se uma matriz tida como representação da cena estática ou quase estática, em que há uma função densidade de probabilidade para cada pixel. A cada nova imagem obtida, utiliza-se alguma distribuição de probabilidade para determinar um valor de intensidade para cada pixel, de forma que estes valores representem o grau de modificação dos píxeis em relação ao fundo. Quanto ao modelo de imagem de fundo, ele pode ser considerado totalmente estático, não sendo atualizado nunca, ou dinâmico, sendo atualizado a uma certa taxa, para levar em conta variações como, pequenos movimentos de objetos que deveriam ser estáticos, variação da iluminação, etc (ZIVKOVIC, 2004). Na Figura 4 está mostrado um exemplo de *background subtraction*.

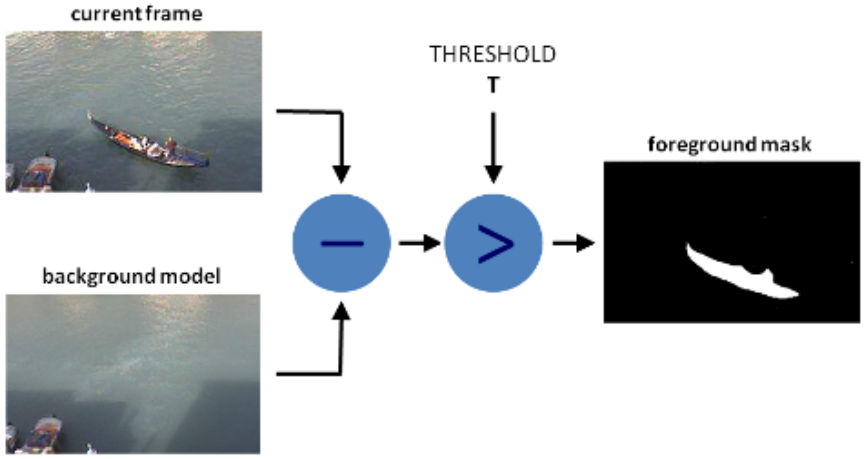

Figura 4: Exemplo de *background* subtraction.
Fonte: OPENCV (2017c).

3.1.2 *BACKGROUND SUBTRACTION* ADAPTATIVO

O OpenCV possui alguns algoritmos já prontos para tal função, que foram testados e apresentaram resultados com altos níveis de ruído, principalmente para mudanças de luminosidade. Para compensar tais mudanças, os algoritmos do OpenCV utilizam um método de adaptação do modelo de imagem de fundo, combinando parcialmente o modelo antigo com as imagens capturadas posteriormente. A adaptação diminui consideravelmente o ruído, entretanto faz com que os alvos sumam gradualmente da imagem caso não se mexam muito.

Por causa das limitações encontradas nos algoritmos do OpenCV, optou-se por desenvolver uma biblioteca de detecção de movimento especificamente para este trabalho, que buscasse atender as suas necessidades. Esta biblioteca foi implementada, em C++, através da classe **MotionDetection**, que está apresentada no apêndice A. O funcionamento do algoritmo é baseado na suposição de que se não há movimentação na imagem,

todos píxeis similares, ou seja, de mesma cor ou de mesma intensidade em escala de cinza, permanecerão similares. Isto é, se não há nenhuma mudança na cena, todos os píxeis continuam com o mesmo valor e, se há mudança apenas de luminosidade, os píxeis passarão a ter um valor maior ou menor. Porém esta variação de cor ocorrerá ao mesmo tempo para todos os píxeis similares. Também, supõe-se que, se há apenas variação de luminosidade, esta variação não deve ocorrer bruscamente, ou seja, diferença de valor de um mesmo píxel em dois quadros subsequentes deve ser pequena.

Com base nestas suposições é possível inferir que, considerando-se todos os píxeis de um certo valor, se apenas um ou alguns poucos variam seu valor bruscamente, isto deve ser uma movimentação na imagem. Por outro lado, se todos os píxeis de um mesmo valor variam pouco e ao mesmo tempo, isto deve ser variação de luminosidade. É a partir destas suposições e conclusões que o algoritmo tira sua ideia central de funcionamento.

O modo de operação do algoritmo é, portanto, primeiramente criar um modelo de imagem de fundo, que é igual à imagem passada a ele como sendo o fundo com objetos estáticos apenas, e, agrupar todos os píxeis de valores similares, através de ponteiros que apontam para um píxel do modelo de fundo (apêndices A.4 e A.9). Entende-se por similar, no algoritmo, píxeis cujo valor seja igual a um valor de referência mais ou menos uma constante de *threshold*, que é definida pelo usuário. A cada nova imagem, é calculado o desvio padrão dos píxeis do modelo de fundo, considerando-se como conjuntos de dados os píxeis de mesmo valor (apêndices A.3 e A.10). É, então, calculado o desvio padrão de cada píxel da nova imagem, tendo como média os valores similares ao valor daquele mesmo píxel no modelo de fundo. Caso a diferença entre o desvio padrão deste novo píxel e o desvio padrão médio dos píxeis similares seja maior do que um *threshold*, definido pelo usuário, ele é considerado como movimentação, pois o seu valor está muito diferente do esperado, e a média de seus similares é estimada como o novo valor de luminosidade do modelo de fundo para este píxel. Caso contrário, a diferença é considerada apenas como variação de luminosidade e seu valor se torna o novo valor do fundo (apêndice A.3). A Figura 5 representa a operação descrita acima. Nota-se que, devido ao grande número de operações iniciais para a geração do modelo de fundo, este algoritmo se torna lento em sua primeira iteração, principalmente para imagens grandes.

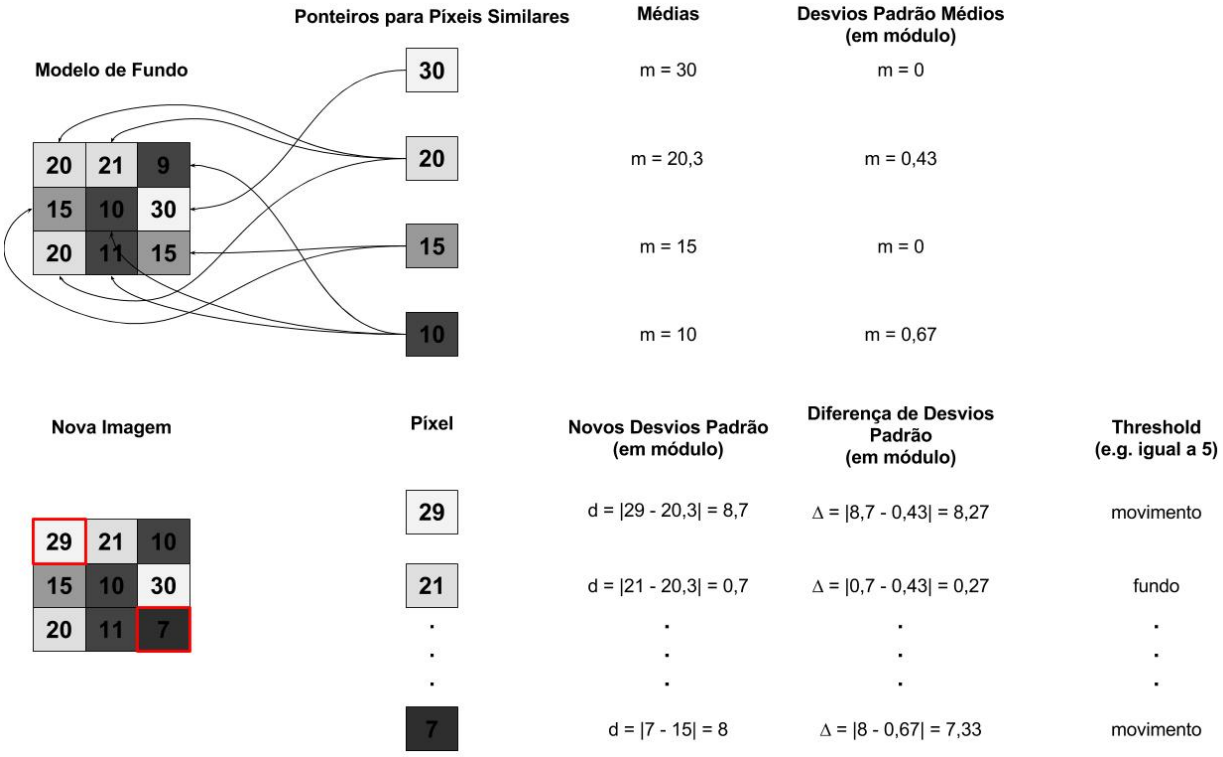

Figura 5: Diagrama da lógica de funcionamento do detector de movimento.
Fonte: Autoria própria.

3.1.3 *THRESHOLDING*

O conceito de *thresholding* é uma transformação binária no qual se escolhe um limite de corte. O método utiliza uma determinada função de *threshold* e compara com os pixels de uma imagem, e determina um novo valor para cada *pixel*, dependendo se o valor do *pixel* for maior ou menor do que o limite. Há diferentes funções para se definir o limite do limiar, como o *threshold* truncado e *threshold* para zero, entretanto o mais simples e intuitivo é o binário, onde o *pixel* ganha valor máximo, branco, se for maior do que o *threshold* e valor mínimo, preto, caso contrário (OPENCV, 2017a). Existem várias formas de se achar o valor de um *pixel*, mas tudo depende da escala usada. A exemplo: escala de cinza, na qual o *pixel* terá um valor entre 0 e 255 para variações de cinza; ou RGB, do inglês *Red - Green - Blue*, no qual o *pixel* terá uma profundidade 3 que irá combinar um valor entre 0 e 255 para cada cor, como exemplo podemos ter um *pixel* com R = 235 + G = 0 + B = 125 que vai resultar na combinação das 3 cores fundamentais para a formação de um valor para aquele *pixel*. Um exemplo de *Threshold* estático é a Figura 6.

Figura 6: Exemplo de *threshold*.
Fonte: OPENCV (2017a).

3.1.4 *DILATION*

A técnica de *dilation*, consiste em dilatar a imagem, ou seja, faz com que regiões com maiores valores de *pixel* cresçam. Isto é feito, convoluindo-se a imagem com uma matriz base. A matriz base é avaliada com relação a cada *pixel* da imagem, sendo que esta matriz também considera os píxeis vizinhos. Por píxeis vizinhos entende-se píxeis abrangidos pela matriz base, o que depende do tamanho desta matriz. O novo valor do *pixel* sendo avaliado será, então, o máximo entre todos os valores de píxeis englobados pela matriz base naquele instante (OPENCV, 2017b). Ou seja, o método faz com que os píxeis de maior brilho sejam replicados para todos os seus píxeis vizinhos, aumentando a área da imagem. As Figuras 7a e 7b representam um exemplo de imagem original e dilatada, respectivamente.

(a) Imagem original (b) Imagem dilatada (c) Imagem erodida

Figura 7: Exemplo de utilização de dilatação e erosão.
Fonte: OPENCV (2017b).

3.1.5 *EROSION*

Em oposição à dilatação, existe a erosã, *erosion*. A operação de erosão ocorre de forma análoga à dilatação, porém, aqui calcula-se o valor mínimo entre os píxeis abrangidos pela matriz base. Desta forma, a imagem obtida após esta operação tem sua área reduzida em relação à original (OPENCV, 2017b). Esta operação é útil para retirar pequenos ruídos da imagem. A Figura 7c é um exemplo de erosão.

3.1.6 CONCLUSÕES SOBRE O TRATAMENTO DE IMAGENS

Tendo em vista a teoria apresentada, a deteção de movimento será realizada de acordo com o diagrama da Figura 8, de forma que esta imagem possa ser processada devidamente. Para implementar tal processamento será utilizada a biblioteca OpenCV, introduzida na Seção 2.2.1.

Nesta aplicação o modelo de *background* será inicialmente estático, devido a sua implementação ser mais simples, e posteriormente migrado para dinâmico, pois a imagem de fundo é de um ambiente aberto, sujeito a movimentação das plantas devido ao vento e variação da luminosidade da cena, principalmente. Por outro lado, o *foreground* que se deseja obter é de animais que possam possivelmente adentrar na armadilha. Conforme indicado na Figura 8, o primeiro passo do algoritmo será transformar a imagem para escalas de cinza, para facilitar o tratamento. Em seguida será realizado o *background* subtraction, utilizando um modelo de background.

O objeto obtido do *background* subtraction, ou seja, o objeto que se moveu na cena *foreground* apresenta muitos ruídos e muitos píxeis que não são totalmente brancos, representando em geral mudanças de luminosidade ou sombras. Para reduzir este problema será utilizado um *threshold* binário, do qual sairá apenas píxeis totalmente brancos ou totalmente pretos.

O uso do *threshold* no entanto, não elimina os pequenos ruídos, deixando a imagem granulada. É preciso passar pela próxima etapa do tratamento, que é a erosão da imagem, como representado na Figura 8. Esta operação removerá a maioria das pequenas aglutinações de píxeis brancos, no entanto diminuirá a área em que realmente houve movimentação. Para compensar isto, chega-se à última etapa, em que a imagem restante com menos ruído é dilatada, abrangendo a maior parte da imagem que realmente se movimentou, sendo esta matriz a saída desejada do tratamento.

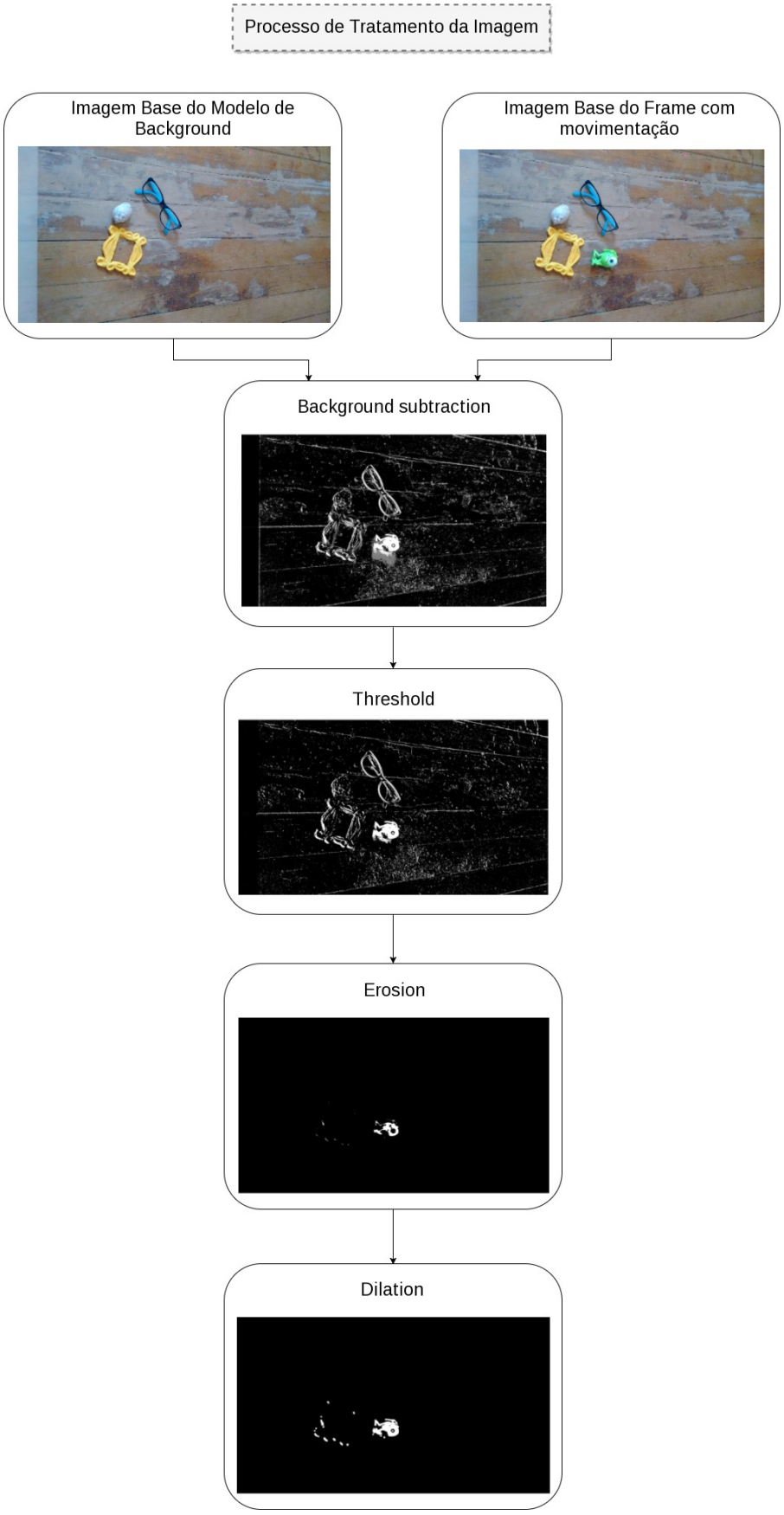

Figura 8: Diagrama do processo de tratamento da imagem.

Fonte: Autoria própria.

Como apresentado na Figura 8, a saída processo de tratamento é uma matriz com píxeis brancos onde houve movimentação e píxeis pretos na área estática. Esta matriz será utilizada para determinar se a movimentação está ocorrendo somente dentro da armadilha, pois ela não pode ser desarmada caso haja animais na fronteira dela ou se a maioria dos animais estiverem fora dela. Também será utilizada para averiguar a área aproximada da armadilha que está ocupada, sendo possível estimar também a quantidade de animais em seu interior.

Para detectar o local onde está ocorrendo movimentação, os dados resultantes do tratamento da imagem serão confrontados com quais píxeis da imagem representam a área interna da armadilha, esses píxeis são fixos, pois sua estrutura e a câmera estão fixas. Este problema trata-se, portanto, de verificar se os píxeis brancos, resultantes do tratamento, são píxeis da área interna, externa ou de fronteira.

A área ocupada da armadilha e a quantidade de animais, por sua vez, podem ser estimados através da contabilização de quantos metros quadrados, aproximadamente, cada *pixel* representa. Este valor, que será fixo, será comparado com o número de píxeis brancos na matriz de saída do processo de tratamento, obtendo-se a área, em metros quadrados, em que houve movimentação. Está área, finalmente, será comparada com a área total da armadilha ou com a área que cada animal ocupa em média.

Realizando-se os procedimentos até aqui citados, obtém-se um armadilha com controle automatizado, capaz de se desarmar quando há movimentação em seu interior. Ela, realizaria o controle através da área movimentada, porém não distinguiria o que causa tal movimentação, desarmando para qualquer tipo de ser vivo ou objeto que adentrasse seus limites.

4 RESULTADOS E DISCUSSÕES

Esta seção apresenta dicussões sobre os códigos desenvolvidos, assim como os testes realizados e os resultados obtidos durante estes testes.

4.1 DOS ALGORITMOS

Com base na biblioteca OpenCV, mencionada na seção 2.2.1, e na metodologia proposta, foram desenvolvidos dois algoritmos. Um algoritmo realiza o *Background Subtraction*, descrito na seção 3.1.1, para detectar movimentos e outro realiza o processamento dos dados de entrada e faz a tomada de decisão sobre o desarme da armadilha. Aqui, o código desenvolvido será apenas citado e sua lógica geral explicada. Os códigos estão integralmente dispostos nos apêndices A, B e C.

4.1.1 CONTROLE DA ARMADILHA

O segundo algoritmo desenvolvido é o responsável por processar os dados extraídos das imagens, de modo a poder realizar alguma ação de controle caso seja necessário. Assim como o anterior, este foi desenvolvido totalmente em C++, também com o auxílio da biblioteca OpenCV. Sua implementação se dá na classe **Trap**, apresentada no apêndice B.

Este código utiliza o algoritmo de detecção de movimento descrito na seção 3.1.2 para realizar o pré-tratamento das imagens da câmera. Na sequência, o algoritmo de controle da armadilha faz um novo tratamento, a extração das informações de movimento e a atuação do sistema quando devido, utilizando-se das regras de negócio definidas para tal. Um diagrama do fluxo do *software* desenvolvido está representado na Figura 9, envolvendo as partes descritas nesta seção quanto as partes da seção 3.1.2. Com o intuito de que o algoritmo possa ser mais genérico, são utilizados parâmetros configuráveis, que podem ser facilmente modificados pelo usuário. As próximas seções tratarão com mais detalhes estas partes do algoritmo.

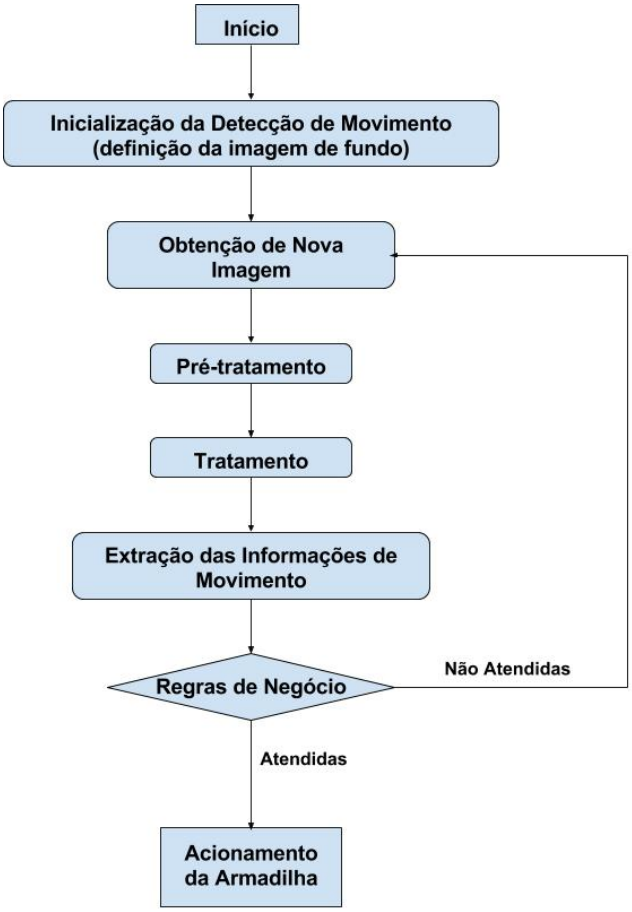

Figura 9: Fluxograma do funcionamento do *software*.
Fonte: Autoria própria.

Para realizar todo o processamento que será descrito na sequência, o algoritmo divide a imagem em duas regiões, ou áreas. Uma externa à armadilha e outra interna a ela, definidas de acordo com as configurações escolhidas pelo usuário, conforme será mais detalhado na seção 4.1.1.5. Esta divisão se dá através de máscaras, em que todos os píxeis que pertencem à área desejada são definidos com valor máximo, 255, e os píxeis que não pertencem à região são definidos com valor mínimo, 0. Quando estas máscaras são confrontadas com a imagem original restam apenas, as regiões de interesse, ou seja, a área externa ou interna. A Figura 10 demonstra tal divisão de áreas.

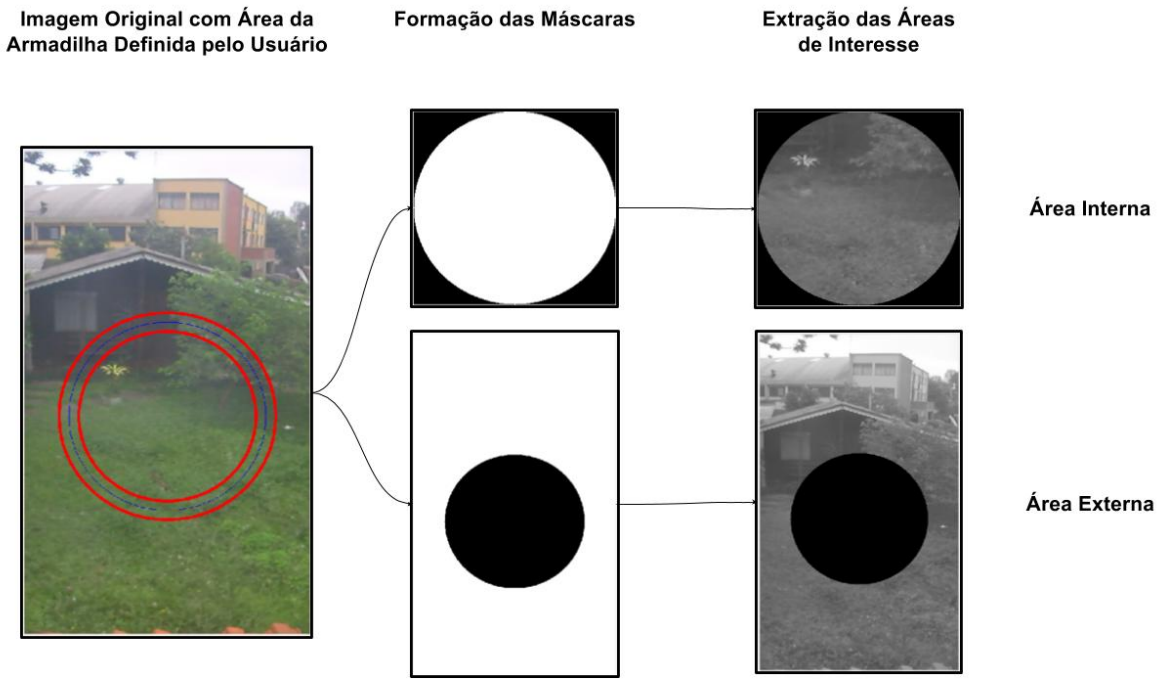

Figura 10: Processo de divisão da imagem em áreas.
Fonte: Autoria própria.

4.1.1.1 TRATAMENTO

Foram desenvolvidos quatro funções para realizar o processo de tratamento. A primeira delas é responsável pela aquisição de imagens e por transformá-las para escala de cinza (apêndice B.12). A segunda tem a funcionalidade de atribuir o referencial de *background*, tanto da área externa quanto interna à armadilha, para a classe **MotionDetection** (apêndice B.17). A terceira realiza a comparação do modelo de *background* com o *frame* atual através da classe **MotionDetection**, extraindo a região movimentada dentro e fora da armadilha. Essa informação de regiões com movimentação é representada através uma máscara de píxeis brancos, com valor 255, que quando aplicada sobre um *frame* resulta em uma imagem na qual apenas as diferenças entre o modelo e a nova imagem são exibidos (apêndice B.4). E, por fim, a quarta destina-se a aplicar os métodos de *erosion* e *dilation*, para reduzir ruídos, também em ambas áreas da armadilha. Além de definir as áreas de exclusão da armadilha, que são as regiões próximas à fronteira da armadilha, onde não pode haver movimentação quando ela for desarmada (apêndice B.15).

4.1.1.2 EXTRAÇÃO DAS INFORMAÇÕES DE MOVIMENTO

Após a imagem ser submetida aos processos de detecção de movimento e filtragem são obtidos *frames*, como o exemplificado na Figura 11, na qual simula-se um movimento qualquer a fim de mostrar que áreas foram movimentadas. Assim como representado na Figura 11, as áreas em branco indicam movimento e as áreas em preto indicam regiões estáticas, ou seja, o resultado do processamento é uma máscara indicando as regiões com movimentação. Note que esta imagem é obtida através do processamento das informações presentes na área interna ou externa da imagem original, que por sua vez são obtidas de conforme explicado na seção 4.1.1 e representado na Figura 10.

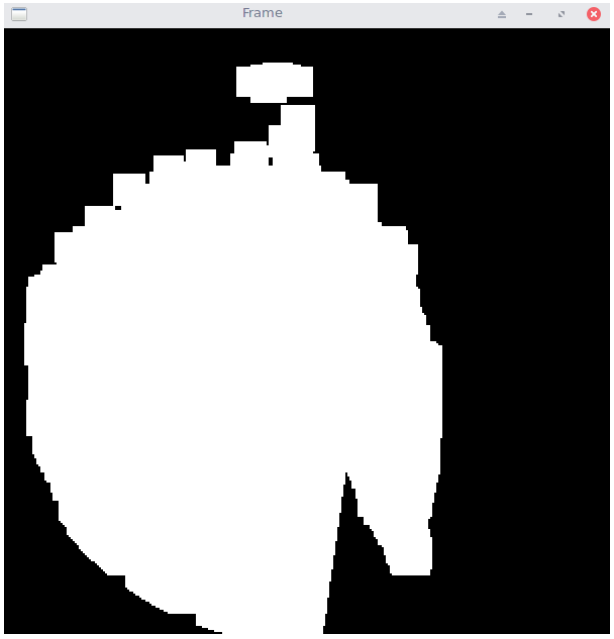

Figura 11: Detalhe de um *frame* submetido ao processo de filtragem.
Fonte: Autoria própria.

As informações mais relevantes para este projeto são o tamanho de cada segmento e sua posição relativa ao centro da armadilha. Essas informações são conhecidas como regiões de interesse ou ROIs, do inglês, *Regions of Interest*. Elas são obtidas a partir da máscara com as regiões movimentadas, com a qual se obtém um vetor de contornos das regiões de movimento, que representam os contornos de potenciais alvos. Com os contornos, é possível calcular a área aproximada, em píxeis, da movimentação, bem como a sua posição na imagem. Em particular, a posição da movimentação na imagem é útil para definir se os alvos estão dentro ou fora da armadilha. O código referente a esta parte do processamento pode ser encontrado no apêndice B.10.

A informação de área, retirada dos contornos, é utilizada tanto para definir o

número aproximado de alvos dentro da armadilha, quanto para definir o que é movimento válido e o que é ruído. O número de indivíduos é obtido dividindo-se a área total movimenta no interior pela área aproximada de um indivíduo médio (apêndice B.5). Enquanto que a distinção entre ruído e movimentação é feita com base na área de referência, definida pelo usuário, que é basicamente a área mínima de um alvo real se movendo e que será abordada na seção 4.1.1.5. Todo o movimento detectado que seja menor do que esta área, isto é, que tenha menos píxeis brancos, de valor 255, é considerado ruído e, portanto, é removido, ou seja, tem seu valor alterado para 0, píxel preto. Sendo assim, restam apenas as movimentações consideradas válidas na máscara (apêndice B.8).

Já a informação de posição dos alvos é utilizada para determinar se o alvo encontra-se dentro ou fora da armadilha, o que serve como base de decisão para outras funções (apêndice B.10). Outra informação com relação à posição, porém extraída de outra forma, é a criação das zonas de exclusão. Estas zonas são criadas também com máscaras, representando anéis concêntricos ao círculo que define a armadilha. A zona externa é definida como um anel com raio interno igual ao raio da armadilha e raio externo 10 porcento maior, configurado inicialmente, do que o raio da armadilha, na qual a porcentagem é definida pelo usuário, como está detalhado na seção 4.1.1.5. A zona interna, por outro lado, possiu raio externo igual ao da armadilha e interno 10 porcento menor que o do armadilha, sendo essa porcentagem o mesmo valor citado para a zona externa (apêndice B.20). A zona de exclusão não permite que a armadilha dispare se houver movimentação dentro delas, ou seja, nas proximidades da fronteira da armadilha, como forma de proteção aos alvos.

Neste ponto, os algoritmos proveem uma solução capaz de ler imagens capturadas por uma câmera, identificar movimentos e classificá-las de acordo com seu tamanho e posição. De posse desses recuros, as implementações discutidas nas próximas seções abordam flexibilização do sistema pela parametrização de configurações e o uso das informações qualitativas para o acionamento e automatização da armadilha.

4.1.1.3 REGRAS DE NEGÓCIO

De posse das informações discutidas nas seções anteriores, como tamanho da área movimentada e sua localização, é possível determinar em que condições a armadilha deve atuar, capturando os animais. Considerou-se também o fato de Javalis serem animais sociais, vivendo em grupos matriarcais de um a seis indivíduos (GRAVES, 1984). Optou-se por permitir que alguns parâmetros, como número mínimo de animais a serem capturados,

fosse configurável, conforme abordado na seção 4.1.1.5.

Com base nestas informações e conforme os objetivos definidos para a armadilha, desenvolveu-se o seguinte conjunto de regras de desarme da armadilha:

1. Não pode haver, em nenhuma hipótese, movimentação nas zonas de exclusão, ou zonas próximas à fronteira, da armadilha;
2. A quantidade de alvos presentes no interior da armadilha deve ser igual ou superior a um limite definido pelo usuário;
3. A quantidade de alvos presentes no exterior da armadilha deve ser igual ou inferior a um limite definido pelo usuário;
4. A armadilha somente pode ser desarmada após todas as condições anteriores terem sido atendidas, ininterruptamente, durante um tempo definido pelo usuário.

As regras 1, 2 e 3, são conferidas com base nas informações descritas na seção 4.1.1.2. As quantidades de alvos, citadas nas regras 2 e 3, são estimativas, realizadas conforme citado também na seção 4.1.1.2. A falha em atender qualquer uma das regras, impede o acionamento da armadilha e, o não atendimento de qualquer uma das três primeiras regras, reinicia o contador de tempo utilizado para a regra 4.

Cada regra foi desenvolvida com base em uma motivação. São motivações, não ferir nenhum animal, para a regra 1; não desarmar a armadilha em um número muito pequeno de indivíduos, deixando a maioria escapar e eliminando um dos grandes objetivos da armadilha, para as regras 2 e 3; não desarmar a armadilha se houver uma grande possibilidade de capturar ainda mais alvos, para as regras 3 e 4; e permitir que os alvos se acomodem dentro da armadilha, reduzindo a atenção deles, por consequência seu tempo de reação e dando tempo para que mais alvos sintam-se confiantes a entrar na armadilha, para a regra 4. A implementação de todas as regras pode ser encontrada no apêndice B.5.

4.1.1.4 ATUAÇÃO DO SISTEMA

Assim como explicitado anteriormente, este projeto tem como foco o desenvolvimento do algoritmo destinado ao controle da armadilha, sem focar em seu protótipo mecânico. Desta forma, a atuação do sistema é traduzida no acionamento de um interruptor eletromecânico: relé.

Como descrito na seção 2.2.4, o sistema atua por meio do acionamento de portas GPIO, presentes no Raspberry. No contexto Linux, cada uma das portas GPIO é descrita como um arquivo, no qual é possível escrever o valor '1' para atribuir nível lógico alto na porta e escrever '0' para atribuir nível lógico baixo. Sendo assim, o algoritmo somente se encarrega de, entregar nível lógico alto para a porta de saída se as regras de negócio forem atendidas, ou de entregar nível lógico baixo caso contrário. O código referente à esta seção pode ser encontrado no apêndice B.1 e no apêndice D.

4.1.1.5 PARÂMETROS CONFIGURÁVEIS

Visto que a aplicação deste trabalho é voltada ao ambiente agrícola, existe uma grande quantidade de variáveis que precisam ser manipuladas para a obtenção dos resultados esperados. Tornando inviável esperar que os valores determinados em ambiente controlado possam ser aplicados em todos os cenários de uso. Visando a adequação do sistema nos mais diversos cenários, esta seção aborda quais parâmetros podem ser ajustados durante a instalação em campo e quais seus impactos.

Desde o ínicio da fase de codificação, a parametrização de configurações e possibilidade de realizar ajustes em tempo real foi foco de atenção. Por isso foi criado um arquivo de configuração, que é passado como argumento para o programa executável, e que permite ao desenvolvedor e usuário alterar o comportamento do sistema sem que seja necessário recompilar todo o código. Este arquivo é livre para ser editado pelo usuário e por meio, dele é possivel configurar:

- **caminho_dev**: Este parâmetro aponta o endereço do *device* Linux a ser usado na captura de imagens, podendo ser uma câmera ou um vídeo. Padrão: "/dev/video0";
- **matriz_erosion**: Determina o tamanho da matriz, conforme citado na seção 3.1.4, a ser utilizada no processo de *erosion* durante a etapa de filtragem. Padrão: 3;
- **matriz_dilation**: Determina o tamanho da matriz, conforme citado na seção 3.1.5, a ser utilizada no processo de *dilation* durante a etapa de filtragem. Padrão: 5;
- **tamanho_alvo**: Determina quantas vezes maior que o *refSize* é o tamanho de um indivíduo médio do objeto alvo da armadilha. Para ajustar esse parâmetro deve-se levar em consideração a área de referência *refSize* calibrada no sistema. Padrão: 3;
- **minimo_dentro**: Específica o número mínimo de alvos dentro da armadilha para que esta seja desarmada. Padrão: 5;

- **maximo_fora**: Específica o número máximo de alvos fora da armadilha para que esta seja desarmada. Padrão: 1;
- **distancia_da_borda**: Variação do raio da armadilha, em porcentagem, para ser considerada como área segura, ou zona de exclusão, conforme discutido nas seções 4.1.1.2 e 4.1.1.3. Padrão: 10;
- **center_x**: Valor, em número de píxeis, na coordenada x da imagem na qual está localizado o centro da armadilha. Padrão: metade da largura da imagem;
- **center_y**: Valor, em número de píxeis, na coordenada y da imagem na qual está localizado o centro da armadilha. Padrão: metade da altura da imagem;
- **radius**: Tamanho do raio da armadilha, em número de píxeis. Padrão: 150;
- **refSize**: Área, em número de píxeis, da referência de área mínima para ser considerada como movimento válido e não como ruído. Padrão: 200;
- **detectThreshold**: Trata-se do parâmetro de utilizado como *threshold* pelo algoritmo de detecção de movimento, conforme descrito na seção 3.1.2. Padrão: 20;
- **detectSimilars**: Refere-se a metade do intervalo de valores píxeis considerados similares, utilizado pelo algoritmo de detecção de movimento, conforme descrito na seção 3.1.2. Padrão: 3;
- **delayToDeploy**: Tempo durante o qual todas as regras de negócio devem ser atendidas antes que a armadilha seja acionada, assim como citado na seção 4.1.1.3. Padrão: 2;
- **caminho_sysconfig**: Caminho para o arquivo de configuração interna do sistema. Padrão: sem caminho padrão.

Apesar de poderem ser definidos manualmente, aconselha-se modificar os parâmetros *center_x*, *center_y*, *radius* e *refSize* por meio das opções de configuração em tempo real disponíveis, cujo modo de utilização pode ser consultado no apêndice E.

Um exemplo desse arquivo, com os valores padrões especificados aqui, pode ser encontrado no apêndice C.

4.2 DOS TESTES E RESULTADOS

Esta seção aborda os testes realizados com o código desenvolvido.

4.2.1 TESTES EM AMBIENTE CONTROLADO

A fim de validar o funcionamento correto da armadilha, foram realizados, primeiramente, testes em ambientes internos e mais controlados. Estes testes foram realizados com a versão inicial do código, para que ficassem claros os pontos que exigiam melhorias.

4.2.1.1 CENÁRIO DO TESTE

Antes da realização dos testes, foi necessário criar um conjunto de testes ao qual o protótipo foi submetido. Como este projeto não visa constuir o aparato mecânico da armadilha, a referência utilizada foi a armadilha *BoarBuster*.

Para determinar o cenário de teste foram utilizados os seguintes dados:

- Tamanho da armadilha: de acordo com o fabricante, a *BoarBuster* possui 5.5 metros de diâmetro;
- Tamanho do Javali: estudos apontam que um javali ou java porco adulto pode alcançar de 125 a 180 cm de comprimento (JÚNIOR et al., 2011). Adotou-se assim um valor médio de 150 cm;
- Tamanho do grupo: segundo GRAVES os grupos podem váriar de 1 a 6 indivíduos. Adotou-se o valor de 3 Javalis como valor mínimo a ser capturado.

Com o intuito de simular a câmera posicionada na parte superior da armadilha, foi utilizado um suporte do tipo Girafa, como mostrado na Figura 12. O suporte foi montado de forma que a câmera ficou a 110 cm do solo e completamente direcionada para baixo: Figura 13.

Figura 12: Detalhe do suporte utilizado para testes.
Fonte: (PLAYTECH, 2016).

Figura 13: Montagem suporte e câmera.
Fonte: Autoria própria.

Para esta configuração, e mais duas alturas diferentes, foi medido o campo de visão da *webcam* Logitech C270 utilizada nos testes. Os resultados estão na Tabela 1.

Configuração	Altura	Comprimento	Largura
1	109 cm	85 cm	62 cm
2	153 cm	120 cm	88 cm
3	182 cm	139 cm	108 cm

Tabela 1: Campo de visão da câmera C270 conforme altura.

Apartir destas medidas e especificações da câmera, foi calculado o valor da abertura vertical da câmera com base equação 1 (PTCOMPUTER, 2015) e (CATELLI et al., 2009).

$$W = DA\frac{1}{fd}. \tag{1}$$

onde W é o campo de visão, D é a distância do objeto, A é a abertura da lente e fd a distância focal.

Na Tabela 2 são mostrados os valores encontrados para cada configuração e o valor médio.

Configuração	Menor Abertura
1	2.28 mm
2	2.30 mm
3	2.37 mm
Média	2.31 mm

Tabela 2: Resultados do cálculo da abertura da lente.

De posse do valor calculado para abertura da lente da câmera em uso, pode-se obter aproximadamente a altura necessária para que, usando a *webcam* C270, fosse possível monitorar uma armadilha com dimensões semelhantes à *BoarBuster*. Tal estimativa está calculada na equação 2.

$$\begin{aligned} 550cm &= D\frac{2.31mm}{4mm}. \\ D &= 952cm. \end{aligned} \tag{2}$$

O resultado obtido na equação (2) de 952 cm, aproximadamente 9,5 metros, é inviável para uma aplicação real. Desse resultado é possivel concluir que a câmera ou seu posicionamento precisam ser alterados. Em primeria análise a melhor solução é buscar no

mercado câmeras com menor distância focal, o que resulta em um campo de visão maior.

Para contornar essa limitação, e considerando que nesta etapa não é possivel identificar javalis por seu biotipo, optou-se por realizar os testes em um ambiente simulado. Como todas as regras de negócio são baseadas em áreas, os testes foram realizados utlizando blocos de papelão de diferentes tamanhos. Sendo um escolhido como referência de área. Deste modo o teste simula a aplicação da armadilha, porém em escala menor.

4.2.1.2 CASOS DE TESTE

Para garantir a usabilidade do protótipo buscou-se casos de teste que refletissem tanto situações de instalação e ajustes da armadilha, como configurar a posição do centro, quanto testes que garantissem a segurança durante o funcionamento, caso de animais em regiões diretamente abaixo da estutura que cerca a armadilha.

Os testes foram realizados em ambiente coberto, porém com janelas, o que resultou em certa variação de luminosidade com o decorrer do tempo. Demais condições como posição do aparato ou regulagem de altura da câmera ficaram inalteradas durante todos os testes.

A seguir é apresentada a lista com uma breve descrição do teste realizado e seu resultado. Todos os testes foram realizados com o software sendo executado no Raspbery Pi via acesso remoto utilizando SSH, sendo todos excutados 5 vezes, exceto quando o teste requisitava mais amostras.

(i) **Reseta-se o sistema. Ao apresentar e configurar uma referência de área o sistema não deve acusar nenhum movimento com área menor que a configurada;**

(ii) **Reseta-se o sistema. O sistema deve permitir configurar o centro da armadilha em qualquer ponto do campo de visão da câmera, alterando o raio quando necessário sem que occora erros na execução. Repetir para 4 posições próximas das bordas e 2 vezes na região central;**

(iii) **Reseta-se o sistema. O sistema deve permitir configurar o raio da armadilha em qualquer ponto do campo de visão da câmera, limitando o raio quando a posição escolhida ultrapasse uma ou ambas das dimensões do campo de visão da câmera sem occora erros na execução. Repetir para 4 posições próximas das bordas e 2 vezes na região central;**

(iv) **Reseta-se o sistema. Parâmetro *tamanho_alvo* = 2. Configura-se o referencial de área. Ao inserir *um* objeto *tamanho_alvo* maior que o referencial de área na região interna da armadilha o sistema deve acusar em linha de comando que existe *um* e apenas *um* alvo válido e zero alvos na região externa;**

(v) **Reseta-se o sistema. Parâmetro *tamanho_alvo* = 2. Configura-se o referencial de área. Ao inserir *um* objeto *tamanho_alvo* maior que o referencial de área na região externa da armadilha o sistema deve acusar em linha de comando que existe *um* e apenas *um* alvo externo e zero alvos na região interna;**

(vi) **Reseta-se o sistema. Parâmetro *tamanho_alvo* = 2. Parâmetro *minimo_dentro* = 3. Parâmetro *maximo_fora* = 0. Configura-se o referencial de área. Ao inserir *X*, alvos no interior da armadilha, sendo *X* menor que *minimo_dentro*, esta não deve desarmar. Repetir para X = 1 e X = 2;**

(vii) **Reseta-se o sistema. Parâmetro *tamanho_alvo* = 2. Parâmetro *minimo_dentro* = 3. Parâmetro *maximo_fora* = 0. Configura-se o referencial de área. Ao inserir *X* alvos no interior da armadilha, sendo *X* maior ou igual que *minimo_dentro*, esta deve desarmar. Repetir para X = 3 e X = 4;**

(viii) **Reseta-se o sistema. Parâmetro *tamanho_alvo* = 2. Parâmetro *minimo_dentro* = 3. Parâmetro *maximo_fora* = 0. Configura-se o referencial de área. Posiciona-se um alvo na região externa. Ao inserir *X* alvos no interior da armadilha, sendo *X* maior ou igual que *minimo_dentro*, esta não deve desarmar. Repetir para X = 3 e X = 4;**

(ix) **Reseta-se o sistema. Parâmetro *tamanho_alvo* = 2. Parâmetro *minimo_dentro* = 3. Parâmetro *maximo_fora* = 2. Configura-se o referencial de área. Posiciona-se três alvos na região interna. Ao inserir *X* alvos no exterior da armadilha, sendo *X* menor ou igual que *maximo_fora*, esta deve desarmar. Repetir para X = 1 e X = 2;**

(x) **Reseta-se o sistema. Parâmetro *tamanho_alvo* = 1. Parâmetro *minimo_dentro* = 1. Parâmetro *maximo_fora* = 0. Parâmetro *distancia_da_borda* = 20. Posiciona-se um alvo na região interna. Configura-se**

o referencial de área. Ao inserir *um* alvo no interior da armadilha, mas estando esse a "0,2 raio unidades de comprimento"da fronteira da armadilha, esta não deve desarmar;

(xi) **Reseta-se o sistema. Parâmetro *tamanho_alvo* = 1. Parâmetro *minimo_dentro* = 1. Parâmetro *maximo_fora* = 1. Parâmetro *distancia_da_borda* = 20. Posiciona-se um alvo na região interna. Configura-se o referencial de área. Ao inserir *um* alvo no exterior da armadilha, mas estando esse a "0.2*raio unidades de comprimento"da fronteira da armadilha, esta não deve desarmar.**

A tabela 3 apresenta o resultado de cada um dos testes realizados assim como observações quanto a alguns resultados.

4.2.2 TESTES EM AMBIENTES EXTERNOS

Assim como discutido na seção 4.2.1, os testes do funcionamento geral e das regras de acionamento da armadilha tiveram resultados satisfatórios como será possível analisar nas imagens seguintes à tabela 3, indicando que a lógica dos algoritmos estão funcionando de acordo com o esperado. O próximo passo foi testar o funcionamento do código em situações mais próximas à sua aplicação, ou seja, envolvendo seres vivos e ambientes naturais, como grama e plantas, no plano de fundo. Como a estrutura para a realização de testes com javalis e de testes com câmeras dispostas da mesma forma que na armadilha não estava disponível para este projeto, optou-se por testar com vídeos gravados em tais ambientes.

Os vídeos utilizados para testes em ambientes externos não foram filmados em uma perspectiva que se adeque ao modelo da armadilha, devido às limitações já citadas. Entretanto este não é um problema para a realização dos testes, visto que o objetivo dos novos testes foi apenas confirmar a funcionalidade do algoritmo para ambientes não controlados e, visto que os testes abordados na seção 4.2.1 já demonstraram o funcionamento em configurações semelhantes às da armadilha. É importante notar ainda, que a qualidade dos resultados depende do ajuste dos parâmetros descritos na seção 4.1.1.5, da maneira mais adequada para cada situação.

Teste	Número de interações	Valor de X	Número de acertos	Número de falhas	Obs
i	5	-	5	0	-
ii	6	-	6	0	-
iii	6	-	6	0	Para ultrapassar a dimensão,de um dos campos de visão pediu-se o raio nas pontas
iv	5	-	4	1	Parte da sombra foi detectada como movimento
v	5	-	5	0	-
vi	5	1	4	1	Quando alvo foi colocado no sentido da fonte de luz, o sistema identificou melhor a sombra, o que diminuiu sua área
	5	2	4	1	Quando alvo foi colocado no sentido da fonte de luz, o sistema identificou melhor a sombra, o que diminuiu sua área
vii	5	3	5	0	-
	5	4	5	0	-
viii	5	3	5	0	-
	5	4	5	0	-
ix	5	1	5	0	A proximidade da sombra dos alvos internos, ocasionou que dois alvos separados fossem considerados juntos.
	5	2	4	1	Sombra de um dos alvos externos foi considera movimento, resultando em 3 alvos externos
x	5	-	5	0	O parâmetro,distancia_da_borda para esse teste foi alterado pra 50, afim de facilitar sua execução
xi	5	-	5	0	O parâmetro,distancia_da_borda para esse teste foi alterado pra 50, afim de facilitar sua execução

Tabela 3: Resultados dos testes de *i* a *xi*.

4.2.2.1 VARIAÇÃO DE LUMINOSIDADE

O primeiro teste visava a averiguação da eficiência do código para a variação de luminosidade. O vídeo gravado consistia de um fundo estático, onde posteriormente era adicionado um objeto móvel, que permanecia o mais parado possível na cena. Então, gradualmente era variada a luminosidade da cena e, em outro momento a luminosidade era variada bruscamente. A Figura 14 mostra a evolução dos frames.

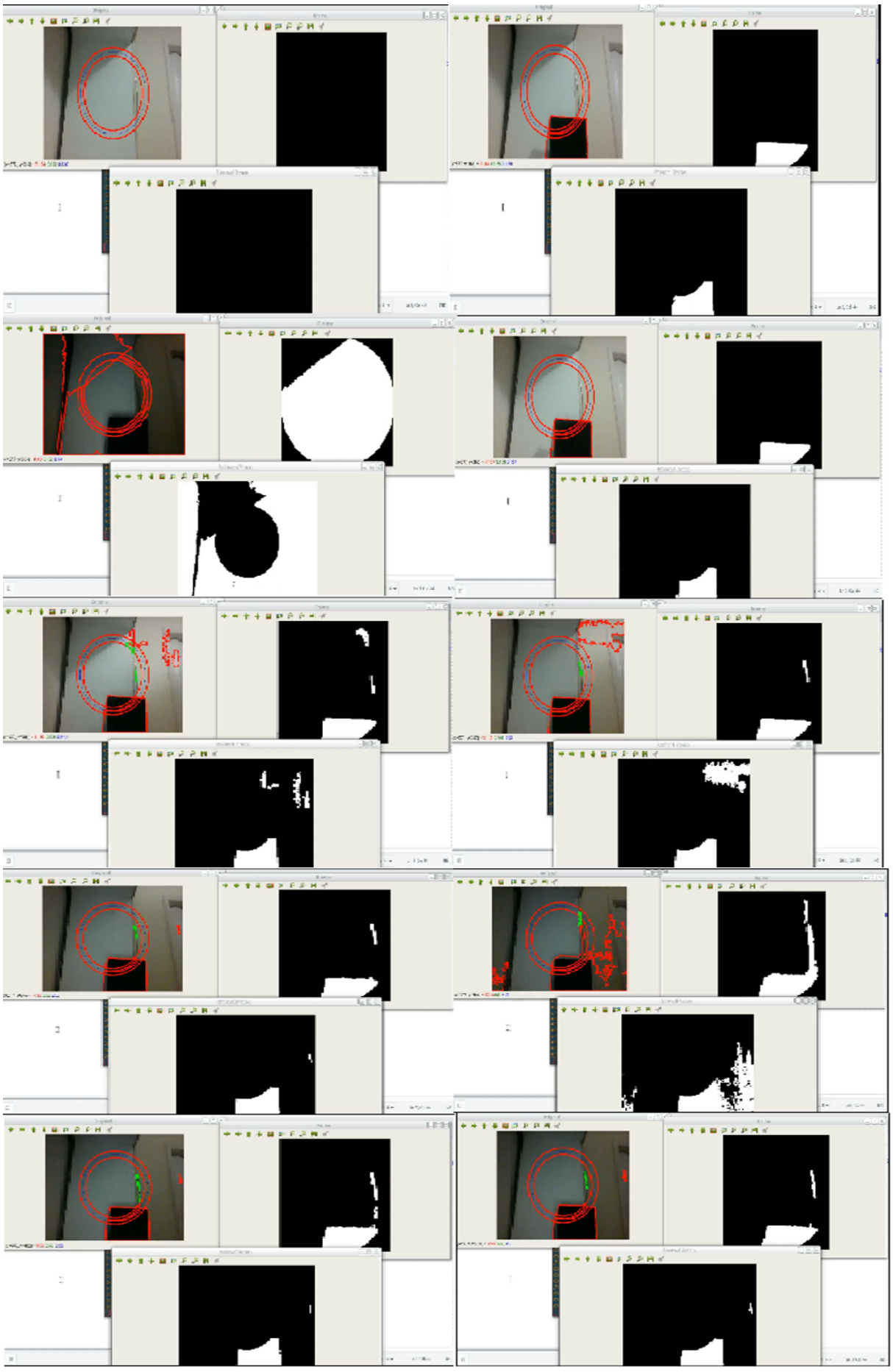

Figura 14: Variação de luminosidade.

Fonte: Autoria própria.

Os resultados obtidos foram que, ao objeto entrar na cena, ele passou a ser detectado como movimento e que, mesmo permanecendo quase estático, ele não foi eliminado pela lógica de adaptação da imagem de fundo. Enquanto a luminosidade era variada, surgiam pequenas áreas classificadas como movimento, que em poucos quadros eram eliminadas pela adaptação do modelo de fundo. Por outro lado, quando a luminosidade foi variada mais bruscamente, toda a imagem foi classificada como movimentação, porém ainda assim, após poucos quadros este ruído foi novamente removido pela adaptação do modelo. Um resultado favorável que pôde ser notado foi que, mesmo havendo adaptação do fundo, o alvo continuou sendo detectado, mesmo tendo permanecido imóvel.

Desta forma, notou-se que o código estava funcionando como o esperado e atendendo à necessidade de sua aplicação, onde a luminosidade irá variar gradualmente durante o dia, devido ao ambiente ser aberto.

4.2.2.2 AMBIENTE COM VEGETAÇÃO NO FUNDO

Conforme o que já foi mencionado, não foi possível filmar um ambiente emglobando apenas um gramado visto de cima, que será o caso da armadilha. Portanto, foi testado o caso onde os alvos estavam se mexendo na grama, porém com árvores e outros objetos no fundo, sendo gravados do alto, com uma perspectiva inclinada. Neste vídeo, três alvos de diferentes tamanhos entraram na cena em uma região onde havia apenas grama alta em seu entorno. Os alvos se movimentavam, entrando, permanecendo e saindo da cena aleatoriamente, como é possível visualizar na Figura 15.

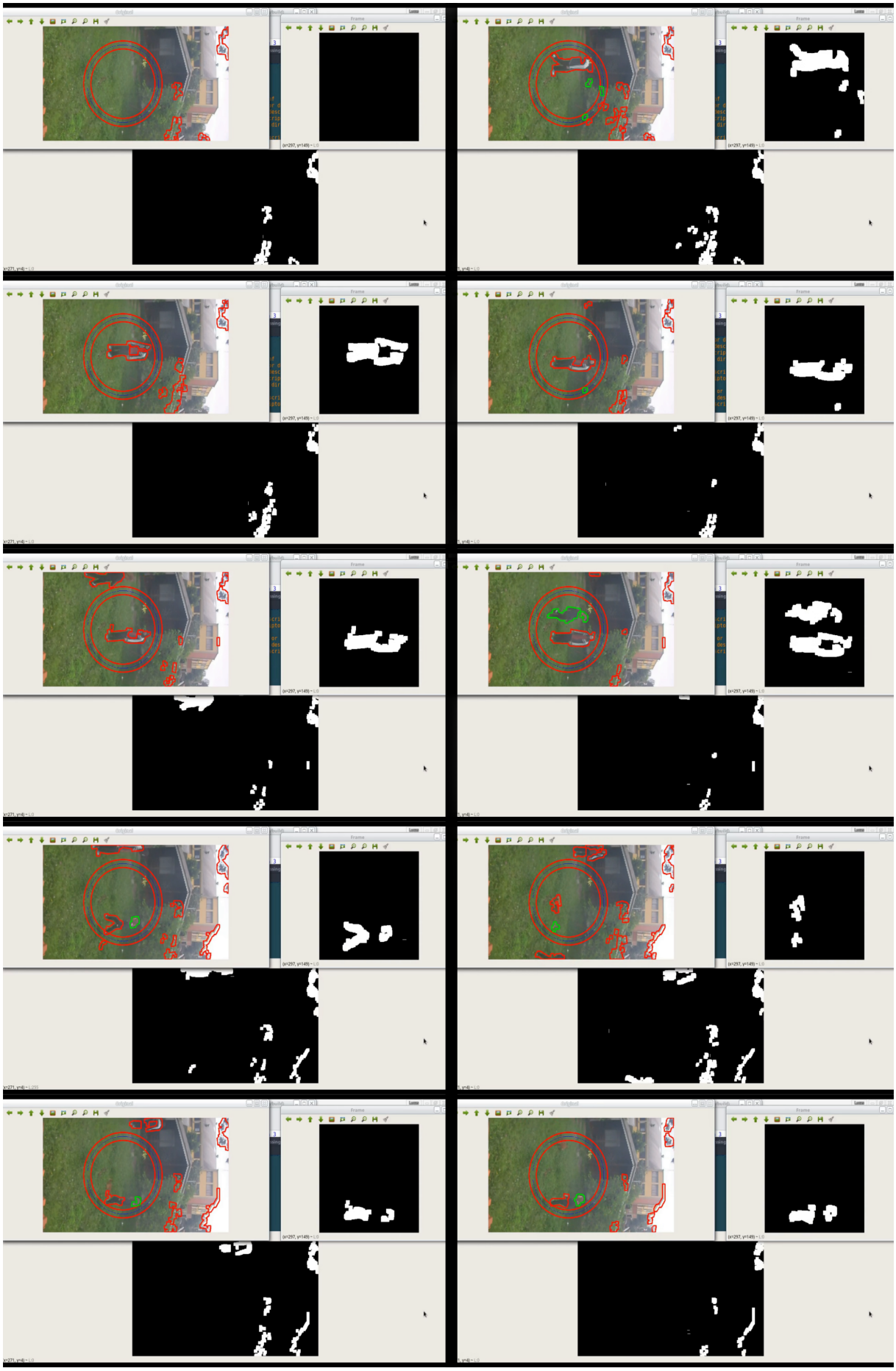

Figura 15: Ambiente não controlado de dia.

Fonte: Autoria própria.

Para esse caso, os resultados mais importantes notados foram que, com os ajustes devidos, o algoritmo foi capaz de localizar a forma completa ou quase completa dos alvos quase que na totalidade do tempo de gravação, não importando se estavam presentes 1, 2 ou os 3 alvos. Também, observou-se que mesmo havendo vento e, portanto, movimentação na grama, ela não foi detectada como movimentação em nenhum momento, excetuando-se uma pequena região, logo após ter sido pisoteada pelos alvos por bastante tempo. Entranto, também notou-se que o algoritmo detectou a movimentação de algumas árvores quando estas eram balançadas pelo vento, o que, no entanto, não representa um problema para o caso de aplicação, visto que na configuração escolhida para a armadilha apenas a grama será filmada.

Este teste também mostra que, com a configuração correta, é possível eliminar o ruído de fundo de ambientes abertos e com grama e, ainda assim, detectar os alvos de forma integral ou quase integral.

4.2.2.3 AMBIENTE COM VEGETAÇÃO NO FUNDO E COM POUCA LUMINOSIDADE

Adicionalmente, também foram testados vídeos públicos de *camera traps*, que são câmeras posicionadas na floresta para capturar imagens noturnas de animais quando estes passam na frente delas. Este tipo de vídeo é gravado com câmeras detectoras de luz infravermelha.

O vídeo utilizado foi gravado em um ambiente com luminosidade bastante reduzida, visto que a luminosidade produzida pelo feixe de luz infravermelha é menor do que a luz natural diurna. Desta forma, a variação de cores no vídeo é menor, sendo que o centro do vídeo apresenta um brilho maior do que resto devido à luz infravermelha focalizada, e a distância de captura de imagens também é menor. Nele, os alvos apareciam em cena, em meio às árvores e se mexiam aleatoriamente, se afastando e aproximando da câmera. É possível observar tais fatos na Figura 16.

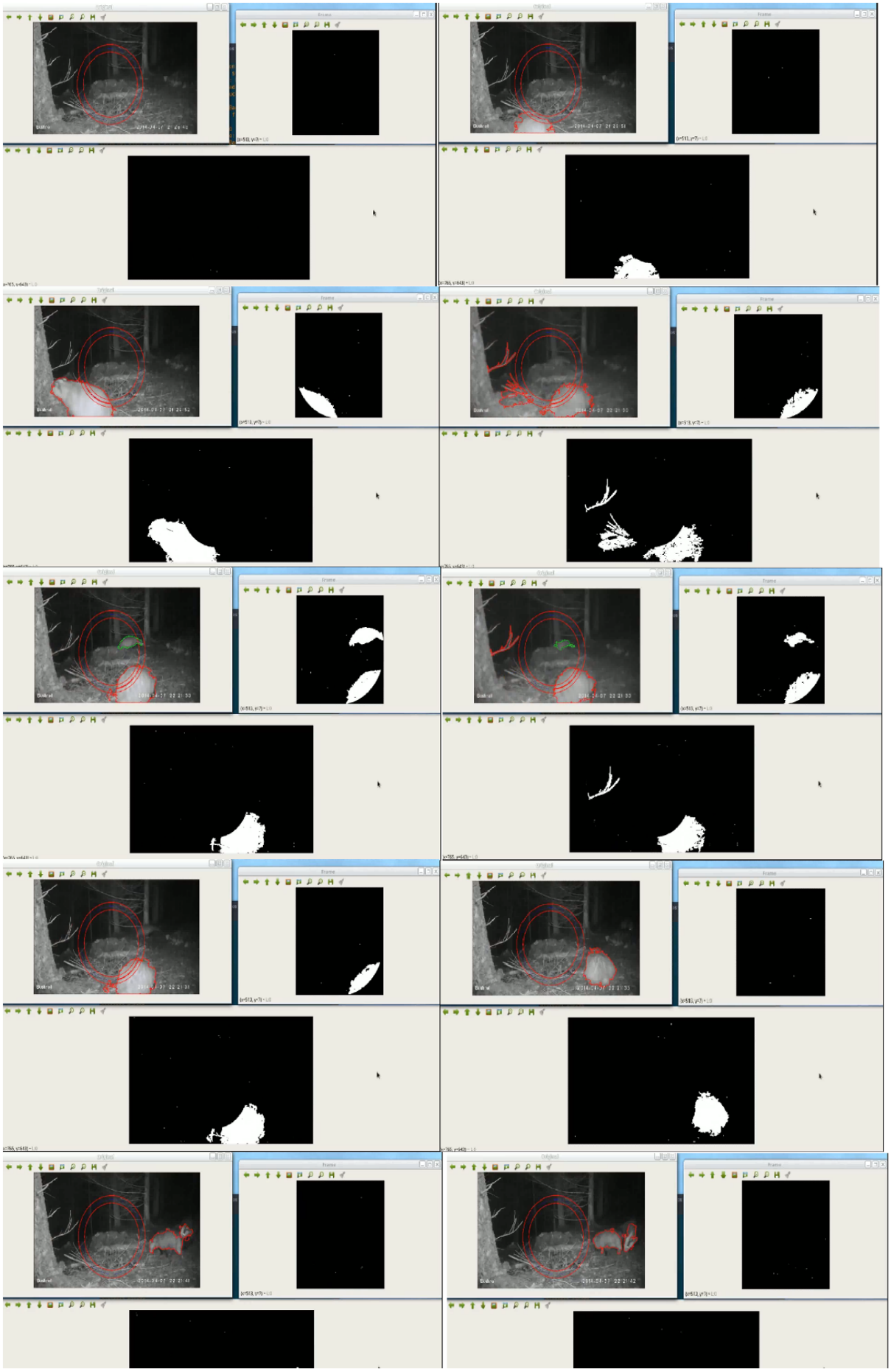

Figura 16: Ambiente não controlado com pouca luz.

Fonte: Autoria própria.

As observações feitas neste caso são semelhantes às descritas na seção 4.2.2.2. Aqui, também foi detectada movimentação das árvores, porém em menor grau, devido ao próprio ambiente mais escuro e de mata fechada. Os alvos foram detectados corretamente, principalmente quando estavam mais próximos à camera e, portanto, mais iluminados, não sendo detectados quando entravam em regiões mais escuras.

Este não é um ambiente planejado para a operação da armadilha a princípio. Entretanto, os resultados mostraram que, utilizando-se uma câmera correta, é possível utilizar o mesmo algoritmo para situações noturnas.

5 CONCLUSÃO

Com objetivo de diminuir o impacto de espécies invasoras sob fauna e flora nativa, assim como prejuízos no setor agropecuário, este projeto buscou desenvolver uma armadilha automatizada capaz de identificar e capturar alvos que entrem em seu espaço de ação, mais especificamente javalis.

Conforme apresentado e discutido na seção 4.2, o sistema desenvolvido cumpre com os requisitos mínimos especificados na seção Objetivos, capítulo 1.4, que são: criar "uma armadilha com acionamento autônomo que captura animais presentes em uma área, evitando causar a estes animais possíveis ferimentos".

Em geral, os testes com alvos vivos, mostraram resultados satisfatórios, em que o sistema é capaz de detectar os alvos, mesmo apresentando poucos ruídos em certas situações. Estes resultados também mostraram que o sistema pode ser utilizado em ambientes externos e menos controlados, sujeito à ação de agentes como o vento. No entanto, realizar a configuração correta dos parâmetros do sistema, nem sempre é uma atividade trivial, sendo necessário testar alguns casos para selecionar o que mais se adequa à uma dada situação.

Porém, para operação ótima, se faz necessário aprimorar alguns processos como o de eliminação de ruídos próximos ao tamanho dos alvos, de formação do modelo de fundo e a velocidade do código. Também é necessário implementar um algoritmo para detectar um alvo específico a partir das informações que já são extraídas neste projeto.

Para trabalhos futuros, sugere-se:

- Utilizar câmeras com maior campo de visão para a realização de testes, como o segundo modelo de câmera proposto nesse trabalho, ou outras disponíveis no mercado, para viabilizar a construção da armadilha;
- Otimizar o código, em especial para plataformas embarcadas;
- Realizar testes com sensores a fim verificar se o desempenho do *software* é incre-

mentado;

- Desenvolver um algoritmo que englobe a capacidade de reconhecimento de alvos específicos, a partir das informações já extraídas com o código aqui desenvolvido.

REFERÊNCIAS

BOARBUSTER. Armadilha boarbuster. 1 imagem, color. 2016. Disponível em: <http://www.boarbuster.com/images/portfolio/trap2.jpg>. Acesso em: 13 out. 2016.

CARRO, L.; WAGNER, F. R. Sistemas computacionais embarcados. **Jornadas de atualização em informática. Campinas: UNICAMP**, 2003.

CATELLI, F.; OURIQUE, P. A.; SILVA, F. S. da. Qual é o "campo de visão" da objetiva de uma câmara fotográfica? **Caderno Brasileiro de Ensino de Física**, v. 26, n. 2, p. 314–327, 2009.

CONVENTION ON BIOLOGICAL DIVERSITY. What are invasive alien species?. 2011. Disponível em: <https://www.cbd.int/invasive/WhatareIAS.shtml>. Acesso em: 25 set. 2016.

GRAVES, H. Behavior and ecology of wild and feral swine (sus scrofa). **Journal of animal science**, The American Society of Animal Science, v. 58, n. 2, p. 482–492, 1984.

HIKVISION. Ir mini bullet network camera. 2017. Disponível em: <http://www.hikvision.com//uploadfile/image/20150911103422996.PDF>. Acesso em: 20 jun. 2017.

IBAMA. Instrução normativa nº 03/2013, de 31 de janeiro de 2013. **Diário Oficial da União República Federativa do Brasil**, Brasília, DF, 01 fev. 2013. Seção 1 p. 88-89, 2013. Disponível em: <http://www.ibama.gov.br/phocadownload/fauna_silvestre_2/legislacao_fauna/2013_ibama_in_003-2013_manejo_javali.pdf>. Acesso em: 29 set. 2016.

JIA, Y. et al. Caffe: Convolutional architecture for fast feature embedding. **arXiv preprint arXiv: 1408.5093**, 2014. Disponível em: <http://caffe.berkeleyvision.org/>. Acesso em: 14 mai. 2017.

JÚNIOR, R. M. P. de O. et al. Ramificações dos nervos supraescapular e subescapulares cranial e caudal em javalis (sus sus scrofa). **Veterinária Notícias**, v. 17, n. 1, p. 62–67, 2011.

KAIZER, M. C.; NOVAES, C. M.; FARIA, M. B. Wild boar sus scrofa (cetartiodactyla, suidae) in fragments of the atlantic forest, southeastern brazil: New records and potential environmental impacts. **Mastozool. Neotrop**, v. 21, p. 343–347, 2014.

KORN, T.; BOMFORD, M. Managing vertebrate pests: Feral pigs. Citeseer, 1996.

LÉVESQUE, O.; BERGEVIN, R. Detection and identification of animals using stereo vision. In: **Proceeedings of Workshop on Visual Observation and Analysis of Animal and Insect Behaviors (held in conjunction with ICPR 2010)**. [S.l.: s.n.], 2010.

LOGITECH. Logitech hd webcam c270 technical specifications. 2014. Disponível em: <http://support.logitech.com/en_us/article/17556>. Acesso em: 3 jun. 2017.

MATHWORKS. Matlab: The language of technical computing. 2017. Disponível em: <https://www.mathworks.com/products/matlab.html>. Acesso em: 29 abr. 2017.

OPEN DETECTION. Opendetection: Introduction. 2017. Disponível em: <http://opendetection.com/introduction_general.html>. Acesso em: 29 abr. 2017.

OPENCV. Basic thresholding operations. 2017. Disponível em: <http://docs.opencv.org/2.4/doc/tutorials/imgproc/threshold/threshold.html>. Acesso em: 14 jun. 2017.

OPENCV. Eroding and dilating. 2017. Disponível em: <http://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html>. Acesso em: 15 jun. 2017.

OPENCV. How to use background subtraction methods. 2017. Disponível em: <http://docs.opencv.org/trunk/d1/dc5/tutorial_background_subtraction.html>. Acesso em: 14 jun. 2017.

OPENCV. Opencv 3.1.0: Open source computer vision. 2017. Disponível em: <http://docs.opencv.org/3.1.0>. Acesso em: 29 abr. 2017.

OPENCV. Opencv: About. 2017. Disponível em: <http://opencv.org/about.html>. Acesso em: 30 abr. 2017.

PEDROSA, F. et al. Current distribution of invasive feral pigs in brazil: economic impacts and ecological uncertainty. **Natureza & Conservação**, Elsevier, v. 13, n. 1, p. 84–87, 2015.

PEDROSA, F. et al. Distribuição de javalis selvagens no brasil. 1 imagem, color. 2015.

PLAYTECH. **181136_Ampliada.jpg**. 2016. Disponível em: <http://www.playtech.com.br/Imagens/produtos/36/181136/181136_Ampliada.jpg>. Acesso em: 17 de ago. 2017.

PTCOMPUTER. **Como calcular o campo de visão para a CCTV**. 2015. Disponível em: <http://ptcomputador.com/Ferragens/computer-peripherals/14677.html>.

SHARMA, S.; SHAH, D. A practical animal detection and collision avoidance system using computer vision technique. **IEEE Access**, IEEE, 2016.

TENSORFLOW. Tensorflow: Getting started. 2017. Disponível em: <https://www.tensorflow.org/get_started/>. Acesso em: 14 mai. 2017.

WALLAU, M. et al. Javali: Qual o tamanho do problema?. **Série Cadernos Técnicos Javalis no Brasil**, n. 3, p. 1–3, mai. 2016.

YU, X. et al. Automated identification of animal species in camera trap images. **EURASIP Journal on Image and Video Processing**, Springer International Publishing, v. 2013, n. 1, p. 52, 2013.

ZIVKOVIC, Z. Improved adaptive gaussian mixture model for background subtraction. In: IEEE. **Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on**. [S.l.], 2004. v. 2, p. 28–31.

APÊNDICE A – CÓDIGO FONTE DA CLASSE MOTIONDETECTION

Neste apêndice estão apresentados os códigos desenvolvidos para a classe **MotionDetection**, responsável pela detecção de movimento.

A.1 ARQUIVO MotionDetector.hpp

```
/*************************************************************

       MotionDectector class definition

              Makes background subtraction in order to detect
       movements. It generates foreground and background
       masks.

*************************************************************/

#ifndef __MOTIONDETECTOR_HPP__
#define __MOTIONDETECTOR_HPP__

//libraries
#include <opencv2/core.hpp>
#include <vector>

//spaces
namespace UMATech
{
    namespace boarstrap
    {
                //class
```

```
class MotionDetector{

        /* Structs used by this class */

        typedef struct{
                //keeps background model
                cv::Mat Image;
                //keeps background mask
                cv::Mat BackgroundMask;
                //keeps foreground mask
                cv::Mat ForegroundMask;
                //keeps pointers to similar pixels
                std::vector< std::vector<uint8_t *> >
                    SimilarityPointer;
        } ImageInfo;

        protected:
                ImageInfo info;

                /* protected functions */
                void findSimilarities(void);
                std::vector<double> standardDeviation(std::vector<
                    uint8_t *> values, double *m);
                double mean(std::vector<uint8_t *> values);
                double mean(std::vector<double> values);
                double module(double value);

        public:
                /* class constructors and destructor */
                MotionDetector(cv::Mat bg);
                MotionDetector(void);
                ~MotionDetector(void);

                /* public functions */
                void setBackground(cv::Mat bg);
```

```
                        void getBackgroundMask(cv::Mat &dest);
                        void getForegroundMask(cv::Mat &dest);
                        void detect(cv::Mat frame, cv::Mat &dest, int
                            threshold = 10, int similarRange = 3);
                };
                //class end
        }
}
//spaces ends

#endif
```

A.2 ARQUIVO MotionDetector.cpp

```
/*************************************************************

        MotionDetector constructors and destructor implementation

*************************************************************/

#include "MotionDetector.hpp"

/*
        Overloaded constructor

                Parameters:
                        -> input cv::Mat bg : image to be considered as
                            background
                Return:
                        -> none

*/
UMATech::boarstrap::MotionDetector::MotionDetector(cv::Mat bg){
        //set the image as background
        setBackground(bg);
```

```
}

/*
        Overloaded constructor

                Parameters:
                        -> none
                Return:
                        -> none

*/
UMATech::boarstrap::MotionDetector::MotionDetector(void){
        //does nothing
}

/*
        destructor

                Parameters:
                        -> none
                Return:
                        -> none

*/
UMATech::boarstrap::MotionDetector::~MotionDetector(void){
        //does nothing
}
```

A.3 ARQUIVO detect.cpp

```
/************************************************************

        detect function implementation

                Detects motion on an image.
```

```
            Parameters:
                  -> input cv:Mat frame : an image in which will
                                 be detected motion
                  -> output cv:Mat dest : a matrix to store the
                                 foreground (motion)
                  -> input int threshold : to separate what will
                                 and what will not be considered motion
                  -> input int similarRange : the range of values
                                 to be considered as similar of a pixel
            Return:
                  -> none

**************************************************************/

#include "MotionDetector.hpp"

void UMATech::boarstrap::MotionDetector::detect(cv::Mat frame, cv::Mat &dest, int
    threshold, int similarRange){
        //msds = mean stardand deviation of similars; m = mean value
        std::vector<double> msds, m;

        //loop to calculate 'msds' and 'm' for each value of a pixel (0 to
            255)
        for(int i = 0; i < 256; i++){
                //auxiliary variable to calculate mean value
                double auxM;
                //auxiliary vector to store the pointers to similar
                   background pixels
                std::vector<uint8_t *> auxS;

                //inferior limit of the range
                int k = i - similarRange;
                if(k < 0)
                        k = 0; //cannot be smaller than 0
```

```
        //loop goes until reach the superior limit of the range or
           the maximum value (255)
        for(; (k <= i + similarRange) && (k < 256); k++)
                auxS.insert(auxS.end(), info.SimilarityPointer[k].begin(), info.
                    SimilarityPointer[k].end());
                //copy to 'auxS' the similiar pixels of value 'k' on
                    the background

        //calculate the standard deviation and the mean, being the
           center value 'i', and stores it on 'msds'
        msds.push_back(mean(standardDeviation(auxS, &auxM)));
        //stores the mean of 'auxS' (similar pixels) on 'm'
        m.push_back(auxM);
}

//iterating over the whole image
for(int i = 0; i < frame.rows; i++){
        for(int j = 0; j < frame.cols; j++){
                //get the value of the pixel on the background model
                uint8_t value = info.Image.at<uint8_t>(i, j);

                //get the value of the pixel on the new frame
                int16_t newPixel = frame.at<uint8_t>(i, j);

                //get the standard deviation of the this pixel in
                    relation to its similars (on the background model
                    )
                double pixelSD = module(newPixel - m[value]);

                //if its standard deviation is greater then the mean
                    standard deviation
                if(module(pixelSD - msds[value]) >= threshold){
                        //it is considered as a motion (white pixel on
                            foreground mask)
```

```
                        info.BackgroundMask.at<uint8_t>(i, j) = 0;
                        info.ForegroundMask.at<uint8_t>(i, j) = 255;
                }
                else{ //otherwise
                        //it is considered as background (white pixel
                            on background mask)
                        info.BackgroundMask.at<uint8_t>(i, j) = 255;
                        info.ForegroundMask.at<uint8_t>(i, j) = 0;
                }

                //update the value of the pixel on model as the mean
                    value of the similar pixels
                info.Image.at<uint8_t>(i, j) = (uint8_t)m[value];
            }
        }

        //return foreground mask
        dest = info.ForegroundMask;
}
```

A.4 ARQUIVO findSimilarities.cpp

```
/*************************************************************

        findSimilarities function implementation

                Create pointers to similar pixels.

                Parameters:
                        -> none
                Return:
                        -> none

*************************************************************/
```

```
#include "MotionDetector.hpp"

void UMATech::boarstrap::MotionDetector::findSimilarities(void){
        info.SimilarityPointer.clear();
        //colours range goes from 0 to 255, so we need 256 pointers
        info.SimilarityPointer.resize(256);

        //iterating over the whole image
        for(int i = 0; i < info.Image.rows; i++){
                for(int j = 0; j < info.Image.cols; j++){
                        //gets pixel value
                        uint8_t val = info.Image.at<uint8_t>(i, j);
                        //gets pixel addres (pointer to the pixel)
                        uint8_t *p = &info.Image.at<uint8_t>(i, j);

                        //store address on the vector of pixels with the same
                            colour
                        info.SimilarityPointer[val].push_back(p);
                }
        }
}
```

A.5 ARQUIVO getBackgroundMask.cpp

```
/*************************************************************

        getBackgroundMask function implementation

                Return the background mask for the last motion
        detection iteration.

                Parameters:
                        -> input/output cv::Mat &dest : matrix to which
                                        the background mask will be copied
                Return:
```

```
                    -> none

**************************************************************/

#include "MotionDetector.hpp"

void UMATech::boarstrap::MotionDetector::getBackgroundMask(cv::Mat &dest){
        dest = info.BackgroundMask;
}
```

A.6 ARQUIVO getForegroundMask.cpp

```
/**************************************************************

        getForeroundMask function implementation

                Return the foreground mask for the last motion
        detection iteration.

                Parameters:
                        -> input/output cv::Mat &dest : matrix to which
                                        the foreground mask will be copied
                Return:
                        -> none

**************************************************************/

#include "MotionDetector.hpp"

void UMATech::boarstrap::MotionDetector::getForegroundMask(cv::Mat &dest){
        dest = info.ForegroundMask;
}
```

A.7 ARQUIVO mean.cpp

```
/************************************************************

        mean function implementation

                Calculates the mean value of a vector.

************************************************************/

#include "MotionDetector.hpp"

/*
        Overloaded function

                Parameters:
                        -> input vector<double> values : vector of
                                        double values
                Return:
                        -> double : the mean of the input values

*/
double UMATech::boarstrap::MotionDetector::mean(std::vector<double> values){
        double mean = 0;

        for(int i = 0; i < values.size(); i++){
                //sum of all values
                mean += values[i];
        }

        //return sum over number of values
        return(mean/values.size());
}

/*
        Overloaded function
```

```
                Parameters:
                        -> input vector<uint8_t *> values : vector
                                        of pointers to values
                Return:
                        -> double : the mean of the input values

*/
double UMATech::boarstrap::MotionDetector::mean(std::vector<uint8_t *> values){
//transforms a vector of pointers into a vector of doubles
        std::vector<double> v;

        for(int i = 0; i < values.size(); i++){
                //get the value of the pixels pointed by the pointers
                v.push_back(*values[i]);
        }

        //call mean function again, but this time passing a vector of
           doubles and return its result
        return(mean(v));
}
```

A.8 ARQUIVO module.cpp

```
/*************************************************************

        module function implementation

                Calculates the module of a value.

*************************************************************/

#include "MotionDetector.hpp"

double UMATech::boarstrap::MotionDetector::module(double value){
        return(value < 0 ? -1*value : value);
```

```
}
```

A.9 ARQUIVO setBackground.cpp

```
/*************************************************************

        setBackground function implementation

                Initialise all image matrices and the vectors of
        similars.

                Parameters:
                        -> input cv::Mat bg : image to be considered as
                                        background model
                Return:
                        -> none

*************************************************************/

#include "MotionDetector.hpp"

void UMATech::boarstrap::MotionDetector::setBackground(cv::Mat bg){
        //copy the background image to the model matrix
        info.Image = bg.clone();
        //make a matrix of ones, with the same size as the model to be the
            background mask
        info.BackgroundMask = cv::Mat::ones(bg.size(), CV_8U);
        //make a matrix of zeros, with the same size as the model to be
           the foreground mask
        info.ForegroundMask = cv::Mat::zeros(bg.size(), CV_8U);
        //create the vectors of similar pixels
        findSimilarities();
}
```

A.10 ARQUIVO standardDeviation.cpp

```
/*************************************************************

        standardDeviation function implementation

                Calculate the standard deviation of a vector of
        values.

                Parameters:
                        -> input vector<uint8_t *> values : vector of
                                        values
                        -> output double *m : the mean of the values
                Return:
                        -> vector<double> : vector with the value of the
                                        standard deviations

*************************************************************/

#include "MotionDetector.hpp"

std::vector<double> UMATech::boarstrap::MotionDetector::standardDeviation(std::
    vector<uint8_t *> values, double *m){
        //store the mean of the values and return it, so we don't need to
            calculate it several times
        *m = mean(values);
        //auxiliary vector to store the standard deviations
        std::vector<double> sd;

        for(int i = 0; i < values.size(); i++){
                //calculate the standard deviation and store it
                sd.push_back(module(*m - *values[i]));
        }

        //return the standard deviations
        return sd;
```

```
}
```

APÊNDICE B – CÓDIGO FONTE DA CLASSE TRAP

Neste apêndice estão apresentados os códigos desenvolvidos para a classe **Trap**, responsável pelo controle da armadilha, bem como o código da função principal do *software*.

B.1 ARQUIVO main.cpp

```
/*************************************************************

       Main program

*************************************************************/

//opencv includes
#include "opencv2/imgcodecs.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/videoio.hpp"
#include <opencv2/highgui.hpp>
#include <opencv2/video.hpp>
#include <opencv2/objdetect/objdetect.hpp>

//standard includes
#include <iostream>
#include <sstream>
#include <fstream>
#include <ctime>

//custom includes
#include "Trap.hpp"
```

```
#include "GPIOClass.h"

/*
       HandleUserInput function definition

              Get the key pressed by the user and handles it.

              Parameters:
                     -> input int input : the code of the pressed key
                     -> input Trap *trap : pointer to the Trap object
              Return:
                     -> none
*/
void HandleUserInput(int input, UMATech::boarstrap::Trap *trap);

/*
       LoadConfigFile function definition

              Load all trap configuration from a file.

              Parameters:
                     -> input const char *path : string containing the
                                  path to the file
                     -> input trap_config_t &config : pointer to the
                                  Trap configuration parameters struct
              Return:
                     -> bool : true if successfully loaded, false
                                  otherwise
*/
bool LoadConfigFile(const char *path, UMATech::boarstrap::trap_config_t &config);

/*
       LoadConfigFile function definition

              Save all trap configuration to a file.
```

```
                Parameters:
                        -> input const char *path : string containing the
                                        path to the file
                        -> input trap_config_t &config : pointer to the
                                        Trap configuration parameters struct
                Return:
                        -> none
*/
void SaveConfigFile(const char *path, UMATech::boarstrap::trap_config_t &config);

/*
        main function implementation

                Unite functions, making the trap control logic.

                Parameters:
                        -> input int argc : number of command line
                                        parameters
                        -> input char **argv : string containing the
                                        command line parameters
                Return:
                        -> int : 0 if everything has gone right
*/
int main(int argc, char **argv)
{
        //variable to store the trap configuration
    UMATech::boarstrap::trap_config_t config;
        //trap object
    UMATech::boarstrap::Trap *trap;
        //gpio object, using port 18
    GPIOClass actuator("18");

        //set gpio port as output
    actuator.setdir_gpio("out");
```

```
    //if a config file has been given and it could be loaded
if(LoadConfigFile(argv[1], config))
{
    std::cout << "Using config file!" << std::endl;
            //initialise the trap object using the loaded parameters
    trap = new UMATech::boarstrap::Trap(config, true);
}
else //otherwise
{
            //initialise the trap object using default parameters
            trap = new UMATech::boarstrap::Trap("/dev/video0", true);
            //get default parameter values
        config = trap->GetConfig();
            //set default video path
            config.devicePath = "/dev/video0";
}

    //discard the first frames, so the camera has time to initialise
       and the code don't process every thing as movement
    trap->DiscardFrames(50);

    //get the video frame rate
    int FPS = trap->GetFPS();

    //start user input as other then the scape key
char input = 'r';

    //start timer
    clock_t startTime = clock();

    //create background model
trap->SetBackground();

    //process frames until a scape key has been pressed
```

```
while(input != 'q' && input != 'Q')
{
        //process user input
    HandleUserInput(input, trap);

        //end timer
        clock_t endTime = clock();
        //time (in seconds) taken processing images
        double seconds = double(endTime - startTime) /
            CLOCKS_PER_SEC;
        //this is how many frames should have been processed
        int framesToDiscard = FPS*seconds;
        //discard past frames, to proccess the current frame
        trap->DiscardFrames(framesToDiscard);

        //start timer again
        startTime = clock();
        //call image processing functions
    trap->ApplyBackground();
    trap->RunImageProcessing();
    trap->ExtractROIs();
    trap->DrawROIs();

        //if the requisites to deploy the trap have been met
    if(trap->DeployTrap())
    {
        std::cout << "Deploy trap!!!" << std::endl;
                //send high level signal to the output port
        actuator.setval_gpio("1");
                //send message to the user, informing the trap has
                    been deployed
                trap->SendUserMessage();
    }
    else //otherwise
    {
```

```
                    //send low level signal to the output port
            actuator.setval_gpio("0");
        }

        //Display image and get user input
        input = trap->DisplayAndGetInput();
    }

        //get configuration parameters again, in case it has been changed
            by the user
    config = trap->GetConfig();

        //save configuration parameters
    SaveConfigFile(argv[1], config);

    delete trap;

    return 0;
}

// HandleUserInput function implementation
void HandleUserInput(int input, UMATech::boarstrap::Trap *trap)
{
    switch(input)
    {
        //Reset parameters to default values and reset background model
        case '0':
        {
            trap->Reset();
            break;
        }
        //Update Background model
        case '1':
        {
            trap->SetBackground();
```

```
        break;
    }
    //Set reference size to the current biggest "movement"
    case '2':
    {
        trap->SetRefSizeByImage();
        break;
    }
    //Set the center of the trap to the center of current biggest "
       movement"
    case '3':
    {
        trap->SetCenterByImage();
        break;
    }
    //Set the radius of the trap as the distance from the center of
       the trap to the center of current biggest "movement"
    case '4':
    {
        trap->SetRadiusByImage();
        break;
    }
    //Set reference size manually
    case '5':
    {
        trap->SetRefSizeManually();
        break;
    }
           //set the center of the trap manually
    case '6':
    {
        trap->SetCenterManually();
        break;
    }
           //set the radius of the trap manually
```

```
        case '7':
        {
            trap->SetRadiusManually();
            break;
        }
        default:
        {
            //Do nothing
            break;
        }
    }
} //main end

// LoadConfigFile function implementation
bool LoadConfigFile(const char *path, UMATech::boarstrap::trap_config_t &config)
{
    bool ret = false;
        //open config file
    std::ifstream config_file(path);
    std::string line;
    std::string token;

    //The config file is expected to have 3 lines:
    //Line 1) Header
    //Line 2) Header of the config by order
    //Line 3) User config separated by comma

        //if the file is opened
    if(config_file.is_open())
    {
        //Skip lines 1 and 2 (just comments)
        std::getline(config_file, line);
        std::getline(config_file, line);
        //Get from third to last line
        //Device path
```

```
std::getline(config_file, token, ',');
config.devicePath = token;
//Erosion matrix size
std::getline(config_file, token, ',');
config.erosionSize = std::stol(token);
//Dilation matrix size
std::getline(config_file, token, ',');
config.dilationSize = std::stol(token);
//Target size
std::getline(config_file, token, ',');
config.refToTargetSize = std::stol(token);
//Minimal number of targets inside
std::getline(config_file, token, ',');
config.minInside = std::stol(token);
//Maximal number of targets outside
std::getline(config_file, token, ',');
config.maxOutside = std::stol(token);
//Frontier distance
std::getline(config_file, token, ',');
config.frontierPercentage = std::stod(token)/100;
//center x coordinate
std::getline(config_file, token, ',');
config.runTimeConfig.center.x = std::stol(token);
//center y coordinate
std::getline(config_file, token, ',');
config.runTimeConfig.center.y = std::stol(token);
//radius
std::getline(config_file, token, ',');
config.runTimeConfig.radius = std::stol(token);
//referenceSize
std::getline(config_file, token, ',');
config.referenceSize = std::stol(token);
//detectThreshold
std::getline(config_file, token, ',');
config.detectThreshold = std::stod(token);
```

```
        //detectSimilars
        std::getline(config_file, token, ',');
        config.detectSimilars = std::stod(token);
        //delayToDeploy
        std::getline(config_file, token, ',');
        config.delayToDeploy = std::stol(token);

                //set parameters as valid
                config.runTimeConfig.valid = true;

        //The last paramater is the path to internal config file
        std::getline(config_file, token, ',');
        config.binConfigPath = token;

                //open binary config file
        std::ifstream bin(config.binConfigPath.c_str());
                //if the file is opened
        if(bin.is_open())
        {
                        //read config struct from the binary file
            bin.read((char *)(&config.runTimeConfig), sizeof(config.runTimeConfig));
            bin.close();
        }
        else //otherwise
        {
            std::cout << "Binary Config not found! Using default." << std::
                endl;
            //config.runTimeConfig.valid is false by default
        }

        config_file.close();
        ret = true;
    }
    else //otherwise
    {
```

```
        std::cout << "Config␣file␣not␣found,␣using␣default␣values" << std::
            endl;
    }
    return ret;
} //LoadConfigFile end

// SaveConfigFile function implementation
void SaveConfigFile(const char *path, UMATech::boarstrap::trap_config_t &config)
{
        //open config file as output
    std::ofstream outfile (path, std::ios_base::out | std::ios_base::trunc);

        //if the file is opened
    if(outfile.is_open())
    {
            //write all parameters to the file, on the expected order
        outfile << "//Arquivo␣de␣configuracao␣do␣usuario,␣altere␣apenas␣a␣
            linha␣3␣respeitando␣a␣seguinte␣formatacao:\n";
        outfile << "//caminho_dev,matriz_erosion,matriz_dilation,
            tamanho_alvo,minimo_dentro,maximo_fora,distancia_da_borda,
            center_x,center_y,radius,refSize,detectThreshold,detectSimilars
            ,delayToDeploy,caminho_sysconfig,\n";
        outfile << config.devicePath;
        outfile << ',';
        outfile << config.erosionSize;
        outfile << ',';
        outfile << config.dilationSize;
        outfile << ',';
        outfile << config.refToTargetSize;
        outfile << ',';
        outfile << config.minInside;
        outfile << ',';
        outfile << config.maxOutside;
        outfile << ',';
        outfile << config.frontierPercentage*100;
```

```
        outfile << ',';
        outfile << config.runTimeConfig.center.x;
        outfile << ',';
        outfile << config.runTimeConfig.center.y;
        outfile << ',';
        outfile << config.runTimeConfig.radius;
        outfile << ',';
        outfile << config.referenceSize;
        outfile << ',';
                outfile << config.detectThreshold;
        outfile << ',';
        outfile << config.detectSimilars;
        outfile << ',';
        outfile << config.delayToDeploy;
        outfile << ',';
        outfile << config.binConfigPath;
        outfile << ',';

        std::cout << "User config saved!" << std::endl;

        outfile.flush();
        outfile.close();
    }
    else //otherwise
    {
        std::cout << "Error saving user config!" << std::endl;
    }

        //open binary config file as output
    std::ofstream binfile(config.binConfigPath.c_str(), std::ios_base::out | std::ios_base::
        trunc);
        //if the file is opend
    if(binfile.is_open())
    {
                //write config struct to it
```

```
        binfile.write((char *)(&config.runTimeConfig), sizeof(config.runTimeConfig));
        std::cout << "Runtime config saved!" << std::endl;
        binfile.flush();
        binfile.close();
    }
    else //otherwise
    {
        std::cout << "Error saving runtime config!" << std::endl;
    }
} //SaveConfigFile end
```

B.2 ARQUIVO Trap.hpp

```
/*************************************************************

       Trap class definition

              Control logic for the trap.

*************************************************************/

#ifndef UMATECH_TRAP
#define UMATECH_TRAP

//opencv includes
#include "opencv2/imgcodecs.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/videoio.hpp"
#include <opencv2/highgui.hpp>
#include <opencv2/video.hpp>
#include <opencv2/objdetect/objdetect.hpp>

//standard includes
#include <iostream>
#include <sstream>
```

```
#include <cmath>

//custom includes
#include "MotionDetector.hpp"

//spaces
namespace UMATech
{
    namespace boarstrap
    {
        //class constants and structures

        //default minimum size to consider as movement
        const double DEFAULT_SIZE = 200;
        //maximum path string length
        const double PATH_MAX_SIZE = 50;

        //struct to define trap position and size
        typedef struct
        {
            cv::Point center;
            bool valid = false;
            double radius = 150;
        } trap_area_t;

        //struct to store image matrices
        typedef struct
        {
            trap_area_t pos; //trap info
            cv::Mat internalMotionMask; //internal motion mask
            cv::Mat greyImage; //input image on grey scale
            cv::Mat colourImage; //input coloured image
            cv::Mat dilatedImage; //input image processed and dilated
            cv::Mat externalProc; //external processed image
            cv::Mat externalMotionMask; //external motion mask
```

```
    cv::Mat greyImageExternal; //external image on grey scale
    cv::Mat inExclusionZone; //internal exclusion zone mask
    cv::Mat outExclusionZone; //external exclusion zone mask
} trap_images_t;

//structure to store information about motion regions
typedef struct
{
    int x;
    int y;
    int nBoars;
    bool inside;
    double size;
    cv::Rect boundRect;
    std::vector<std::vector<cv::Point> > contour;
    double distanceFromCenter;
} roi_info_t;

//structure with default config values
typedef struct
{
    int minInside = 5; //min number of targets inside
    int maxOutside = 1; //max number of targets outside
    int erosionSize = 3; //erosion matrix size
    int dilationSize = 5; //dilation matrix size
    int detectThreshold = 20; //MotionDection class parameter
    int detectSimilars = 3; //MotionDection class parameter
    std::string devicePath; //video device path
    std::string binConfigPath; //binary configuration file path
    trap_area_t runTimeConfig; //trap info
    double refToTargetSize = 3; //refToTargetSize*referenceSize = the
        median size of a target
    double frontierPercentage = 0.1; //distance to trap border
    double referenceSize = DEFAULT_SIZE; //the minimum size to
        consider as movement
```

```
    double delayToDeploy = 2; //time in seconds to delay the deploy
} trap_config_t;


class Trap
{
    private:
        /* private variables */
        int _keyboard;
        trap_images_t _img; //image matrices
        cv::Point _poiOffset;
        trap_config_t _config; //configuration parameters
        cv::VideoCapture _video; //video object
        std::vector<roi_info_t> _roi; //vector of regions of interest
        UMATech::boarstrap::MotionDetector _internal; //motion detector
            for internal region
        UMATech::boarstrap::MotionDetector _external; //motion detector
            for external region

        const char *_windowFrame = "Frame";
        const char *_windowOriginal = "Original";
        const char *_windowFrameExternal = "External Frame";

        bool _windowEnable;
        bool _useFullImage = false;
        bool _canDeploy = false;

        int _maxAreaIdx = -1;
        int _validMovements = 0;
        int _invalidMovements = 0;

        int _areaInside = 0;
        int _areaOutside = 0;

        //timer to deploy the trap
```

```
    clock_t _startTime;

    /* private functions */
    int GetMaxAreaIdx(void);
    void GetCurrentFrame(void);
    cv::Point TrapArea(trap_images_t &img);
    cv::Point trap_area(trap_images_t &img);
    void ExtractROIs(cv::Mat &src, bool inside);
    void RunImageProcessing(cv::Mat &MOG2Mask, cv::Mat &dest);

public:
    /* constructors and destructors */
    ~Trap(void);
    Trap(trap_config_t &config, bool showWindows);
    Trap(const char *dev, bool showWindows = true);

    /* public functions */
    void Reset(void);
    void SetRadiusByImage(void);
    void SetRadiusManually(void);
    void SetCenterByImage(void);
    void SetCenterManually(void);
    void SetRefSizeByImage(void);
    void SetRefSizeManually(void);
    bool DeployTrap(void);
    void ExtractROIs(void);
    void SetBackground(void);
    void ApplyBackground(void);
    void RunImageProcessing(void);
    void FitTrapArea(void);
    void DiscardFrames(int n);
    bool ExclusionZoneClear(void);
    void SetExclusionZone(void);
    void SendUserMessage(void);
```

```
            void DrawROIs(bool drawCenter = false);
            char DisplayAndGetInput(bool block = false);

            int GetValidMovements(void){return _validMovements;};
            std::vector<roi_info_t> *GetROIs(void){return &_roi;};
            int GetInvalidMovements(void){return _invalidMovements;};
            double GetRefSizeValue(void){return _config.referenceSize;};
            void SetErosionSize(int size) { _config.erosionSize = size;};
            void SetDilationSize(int size) { _config.dilationSize = size;};
            double GetTargetMultipler(void){return _config.refToTargetSize;};
            void SetTargetMultipler(double value){_config.refToTargetSize = value
                ;};
            trap_config_t GetConfig(void){_config.runTimeConfig = _img.pos;
                return _config;};
            double GetMaxAreaValue(void){return _maxAreaIdx != -1 ?_roi[
                GetMaxAreaIdx()].size : 0.0f;};
            int GetFPS(void){return _video.get(cv::CAP_PROP_FPS);};
        }; //class Trap
    } //namespace boarstrap
} //namespace UMATech

#endif //UMATECH_TRAP
```

B.3 ARQUIVO Trap.cpp

```
/*************************************************************

        Trap constructors and destructor implementation

*************************************************************/

#include "Trap.hpp"

/*
        Overloaded contructor
```

```
                Parameters:
                        -> input const char *dev : string containing the
                                        path to video device
                        -> input bool showWindows : true to show image
                                        windows, false otherwise
                Return:
                        -> none
*/
UMATech::boarstrap::Trap::Trap(const char *dev, bool showWindows) : _video(dev)
{
        //if video device is not opened
    if(!_video.isOpened())
    {
        std::cout << "Error opening dev!: " << dev << std::endl;
                //die on error
        throw("Failed to open device!");
    }
    if(showWindows) //otherwise
    {
        //Create windows (600x600) to show original and processed images
        cv::namedWindow(_windowOriginal, cv::WINDOW_NORMAL); //original
        cv::namedWindow(_windowFrame, cv::WINDOW_NORMAL); //internal
            motion mask
        cv::namedWindow(_windowFrameExternal, cv::WINDOW_NORMAL); //
            external motion mask
        cv::resizeWindow(_windowOriginal, 600, 600);
        cv::resizeWindow(_windowFrame, 600, 600);
        cv::resizeWindow(_windowFrameExternal, 600, 600);

        std::cout << "Windows Created!" << std::endl;
    }
    _windowEnable = showWindows;
}
```

```
/*
        Overloaded contructor

                Parameters:
                        -> input trap_config_t &config : configuration
                           parameters
                        -> input bool showWindows : true to show image
                                    windows, false otherwise
                Return:
                        -> none
*/
UMATech::boarstrap::Trap::Trap(trap_config_t &config, bool showWindows) : Trap(
    config.devicePath.c_str(), showWindows)
{ //call the other trap constructor
        //set configuration parameters
    _config = config;
    if(_config.runTimeConfig.valid)
    {
        _img.pos = _config.runTimeConfig;
    }
}

/*
        Destructor

                Parameters:
                        -> none
                Return:
                        -> none
*/
UMATech::boarstrap::Trap::~Trap(void)
{
    _video.release();
}
```

B.4 ARQUIVO ApplyBackground.cpp

```
/************************************************************

        ApplyBackground function implementation

                Get new frame, get region masks and get motion
        regions.

                Parameters:
                        -> none
                Return:
                        -> none

************************************************************/

#include "Trap.hpp"

void UMATech::boarstrap::Trap::ApplyBackground(void)
{
        //get new frame
    GetCurrentFrame();
        //get masks and masks offset
    _poiOffset = TrapArea(_img);

        //motion detection for internal region
        _internal.detect(_img.greyImage, _img.internalMotionMask, _config.
            detectThreshold, _config.detectSimilars);
        //motion detection for external region
        _external.detect(_img.greyImageExternal, _img.externalMotionMask, _config.
            detectThreshold, _config.detectSimilars);
}
```

B.5 ARQUIVO DeployTrap.cpp

```
/*************************************************************

        DeployTrap function implementation

                Avaliate deploy conditions (rules).

                Parameters:
                        -> none
                Return:
                        -> bool : true if conditions have been met (can
                                        deploy trap), false otherwise

*************************************************************/

#include "Trap.hpp"

bool UMATech::boarstrap::Trap::DeployTrap(void)
{
        /** 1st condition -> no movement on exclusion zone **/

        if(!ExclusionZoneClear()){
                _canDeploy = false;
                //if there is movement, return false
                return false;
        }

        /** 2nd condition -> minimum number of individuals inside the trap
            **/

        //approximated number of individuals inside the trap
        double nInside = _areaInside/(_config.refToTargetSize*_config.referenceSize);
        printf("%lf inside\n", nInside);
        if(nInside < _config.minInside){
                _canDeploy = false;
```

```
        //if there are not enough indiviuals inside the trap, return
            false
        return false;
}

/** 3rd condition -> maximum number of individuals outside the
    trap **/

//approximated number of individuals outside the trap
double nOutside = _areaOutside/(_config.refToTargetSize*_config.referenceSize)
    ;
printf("%lf outside\n", nOutside);
if(nOutside > _config.maxOutside){
        _canDeploy = false;
        //if there are too many indiviuals outside the trap, return
            false
        return false;
}

/** 4th condition -> time with the conditions being satisfied **/

if(!_canDeploy){
        _canDeploy = true;
        //start timer if all conditions are satisfied and the timer
            has not been started yet
        _startTime = clock();
}
else{ //if the timer has already been started
        //check how many seconds have been elapsed
        clock_t endTime = clock();
        double seconds = double(endTime - _startTime) /
            CLOCKS_PER_SEC;
        if(seconds < _config.delayToDeploy)
                return false; //if has not reached the delay time,
                    return false
```

```
                    //Do not turn '_canDeploy' false (this is job for the
                        other conditions), otherwise the trap will never
                        be deployed
        }

        //if all conditions where satisfied, return true
        return true;
}
```

B.6 ARQUIVO DiscardFrames.cpp

```
/*************************************************************

        DiscardFrames function implementation

                Discard a given number of frames from the video or
        camera input.

                Parameters:
                        -> intput int n : number of frames to be
                                        discarded
                Return:
                        -> none

*************************************************************/

#include "Trap.hpp"

void UMATech::boarstrap::Trap::DiscardFrames(int n)
{
        for(int i = 0; i < n; i++){
                //get frames and do nothing
                GetCurrentFrame();
        }
}
```

B.7 ARQUIVO DisplayAndGetInput.cpp

```
/************************************************************

        DisplayAndGetInput function implementation

                Display the images in windows and get the code of
        the key pressed by the user, if any.

                Parameters:
                        -> intput bool block : if true, get user input
                                        as a blocking operation, if false get
                                        user input as non-blocking operation
                Return:
                        -> char : code for the pressed key

************************************************************/

#include "Trap.hpp"

char UMATech::boarstrap::Trap::DisplayAndGetInput(bool block)
{
        //store the key code
    char ret;

        //if showing windows is enabled
    if(_windowEnable)
    {
                //show coloured original image
        cv::imshow(_windowOriginal, _img.colourImage);
                //show the processed image for internal region
        cv::imshow(_windowFrame, _img.dilatedImage);
                //show the processed image for external region
        cv::imshow(_windowFrameExternal, _img.externalProc);
    }
```

```
        //if should be a blocking operation
    if(block)
    {
                //wait until the user press a key
        ret = cv::waitKey(0);
    }
    else //otherwise
    {
                //wait for 100 milliseconds only
        ret = cv::waitKey(100);
    }

        //return the code for the pressed key
    return ret;
}
```

B.8 ARQUIVO DrawROIs.cpp

```
/*************************************************************

        DrawROIs function implementation

                Draw the regions of interest and remove noise.

                Parameters:
                        -> intput bool drawCenter : draw a circle
                                        representing the center of the biggest
                                        moved area, if true
                Return:
                        -> none

*************************************************************/

#include "Trap.hpp"
```

```
void UMATech::boarstrap::Trap::DrawROIs(bool drawCenter)
{
    //Reset movements counters
    _invalidMovements = 0;
    _validMovements = 0;

    //Only areas that are bigger than the reference area will be draw
    //and the biggest area will be diplayed in Red, others in green
        //Only draw if windows are enabled
    for(size_t i = 0; _windowEnable && i < _roi.size(); i++)
    {
                //if the movement is bigger than the minimum size of a
                    movement
                if(_config.referenceSize < _roi[i].size)
        {
                //As the ROIs are calculated refereing to a region of the
                    orignal
                //image, not the image it self, we have to compensated to
                    the offset
                //in order to plot the ROI in the desired area
                _roi[i].boundRect.x += _useFullImage ? 0 : _poiOffset.x;
                _roi[i].boundRect.y += _useFullImage ? 0 : _poiOffset.y;
                _roi[i].boundRect.width = (_roi[i].boundRect.x + _roi[i].boundRect.
                    width) > _img.colourImage.cols ? (_img.colourImage.cols - _roi[i].
                    boundRect.x) : _roi[i].boundRect.width;
                _roi[i].boundRect.height = (_roi[i].boundRect.y + _roi[i].boundRect.
                    height) > _img.colourImage.rows ? (_img.colourImage.rows - _roi[i].
                    boundRect.y) : _roi[i].boundRect.height;

                        //if it is inside the trap
                        if(_roi[i].inside){
                        //Is a valid movement
                        _validMovements++;

                                //if this is the biggest area
```

```
            if(i == GetMaxAreaIdx())
            {
                    //draw red contour around the object
                    cv::drawContours(_img.colourImage, _roi[i].
                        contour, -1, cv::Scalar(0, 0, 255), 4, 8, cv::
                        noArray(), INT_MAX, _poiOffset);
                if(drawCenter)
                {
                    //If draw center is true, we also plot the
                        center of rectangle of the biggest
                        area
                    std::cout << "Center: X = " << _roi[i].
                        boundRect.x << " Y = " << _roi[i].
                        boundRect.y << std::endl;
                    cv::Point tmp(_roi[i].boundRect.x + _roi[i].
                        boundRect.width/2, _roi[i].boundRect.y + _roi[
                        i].boundRect.height/2);
                    circle(_img.colourImage, tmp, 5, cv::Scalar(0, 0,
                        255));
                }
            }
            else if(!drawCenter)
            {
                    //draw green contour around the object
                    cv::drawContours(_img.colourImage, _roi[i].
                        contour, -1, cv::Scalar(0, 255, 0), 4, 8, cv::
                        noArray(), INT_MAX, _poiOffset);
            }
            else
            {
                //Do nothing
                    //draw only the biggest region if '
                        drawCenter' is enabled
            }
        }
```

```
                    else{ //if it is outside the trap
                            //draw red contour around the object
                            cv::drawContours(_img.colourImage, _roi[i].contour, -1,
                                cv::Scalar(0, 0, 255), 4);
                    }
        }
        else //if is smaller than the minimum valid movement area
        {
            //Movement Count as invalid, so treat it as noise and remove it
            _invalidMovements++;

                    //create a zero filled matrix, the same size of the
                       movement
                    cv::Mat aux = cv::Mat::zeros(_roi[i].boundRect.size(), CV_8U);

                    if(_roi[i].inside){ //if it is inside the trap
                            //copy the zero matrix to the moved area on
                               internal region
                            //i.e., remove it, turning it black
                            aux.copyTo(_img.dilatedImage(_roi[i].boundRect));
                    }
                    else{ //if it is outside the trap
                            //same for external region
                            aux.copyTo(_img.externalProc(_roi[i].boundRect));
                    }
        }
    }
}
```

B.9 ARQUIVO ExclusionZoneClear.cpp

```
/*********************************************************

      ExclusionZoneClear function implementation
```

```
            Check if there is no movement inside the exclusion
      zone.

            Parameters:
                  -> none
            Return:
                  -> bool : true if there is no movement inside
                              the trap (i.e. it is clear), false
                              otherwise

*************************************************************/

#include "Trap.hpp"

bool UMATech::boarstrap::Trap::ExclusionZoneClear(void){
      //mask of movement inside the inner exclusion zone
      cv::Mat auxIn = _img.inExclusionZone & _img.dilatedImage;
      //mask of movement inside the outter exclusion zone
      cv::Mat auxOut = _img.outExclusionZone & _img.externalProc;

      //count number of non zero pixels inside exclusion zones (i.e.
         moved pixels)
      int whiteAreaIn = cv::countNonZero(auxIn);
      int whiteAreaOut = cv::countNonZero(auxOut);

      //if there is no movement (white pixels) on exclusion zones
      if((whiteAreaIn == 0) && (whiteAreaOut == 0))
            return true;
      else //if there is movement
            return false;
}
```

B.10 ARQUIVO ExtractROIs.cpp

```
/*************************************************************
```

```
        ExtractROIs function implementation

                Extract ROI information from motion masks.

*************************************************************/

#include "Trap.hpp"

/*
        Overloaded function

                Parameters:
                        -> input cv::Mat &src : matrix containing a
                                        motion mask
                        -> input bool inside : true if it is about the
                                        region inside the trap
                Return:
                        -> none
*/
void UMATech::boarstrap::Trap::ExtractROIs(cv::Mat &src, bool inside)
{
    std::vector<std::vector<cv::Point> > contours;
    //At this point all "movements" are represented as white areas
    //in the image. This process will set a contour around all areas
    //that have been changed when compared to the background model.
    cv::findContours(src, contours, cv::RETR_EXTERNAL, cv::
        CHAIN_APPROX_SIMPLE);


    std::vector<std::vector<cv::Point> > contoursPoly(contours.size());
    std::vector<roi_info_t> tmp(contours.size());

    //A new set of ROIs will be defined, so we reset the _maxAreaIdx
    //but only for the inside region
```

```
if(inside)
{
    _maxAreaIdx = -1; //id of biggest movement area
}

    //iterate over all regions
for(size_t i = 0; i < contours.size(); i++)
{
    tmp[i].inside = inside;
    //In order to simplifly futher calculations we optimize the
       contours with approxPolyDP
    //reducing the number of vertices
    cv::approxPolyDP(cv::Mat(contours[i]), contoursPoly[i], 3, true);

    //The last step is to transform this areas into rectangles for
    //trap calibration by image
            tmp[i].contour.push_back(contoursPoly[i]);
    tmp[i].boundRect = cv::boundingRect(cv::Mat(contoursPoly[i]));

    //The area value is based on the contour, not the rectangle,
    //minimizing the approximation error
    tmp[i].size = contourArea(contoursPoly[i]);
            //if it as a valid movement (bigger than the reference size)
            if(_config.referenceSize < tmp[i].size){
                    if(inside){
                            //increase the area moved inside the trap
                            _areaInside += tmp[i].size;
                    }
                    else{
                            //increase the area moved outside the trap
                            _areaOutside += tmp[i].size;
                    }
            }
```

```
        //We can also use the moments to calculate the center of the
            contour
        cv::Moments M = cv::moments(contoursPoly[i]);
        tmp[i].x = inside == true ? (int)M.m10/M.m00 + _poiOffset.x : (int)M.m10/
            M.m00;
        tmp[i].y = inside == true ? (int)M.m01/M.m00 + _poiOffset.y : (int)M.m01/
            M.m00;

        //And from it the distance from the center, this information can
            be used
        //to determine if the ROIs are in safe distance from the img edges
        tmp[i].distanceFromCenter = (_img.pos.center.x - tmp[i].x) * (_img.pos.center.
            x - tmp[i].x);
        tmp[i].distanceFromCenter += (_img.pos.center.y - tmp[i].y) * (_img.pos.
            center.y - tmp[i].y);
        tmp[i].distanceFromCenter = std::sqrt(tmp[i].distanceFromCenter);
    }

        //store roi information
    _roi.insert(_roi.end(), tmp.begin(), tmp.end());
}

/*
        Overloaded function

                Parameters:
                        -> none
                Return:
                        -> none
*/
void UMATech::boarstrap::Trap::ExtractROIs(void)
{
        //clear previous roi information
    _roi.clear();
        //clear previous moved areas information
```

```
        _areaInside = 0;
        _areaOutside = 0;
        //call the other function to process internal ROIs
    ExtractROIs(_img.dilatedImage, true);
        //call the other function to process external ROIs
    ExtractROIs(_img.externalProc, false);
}
```

B.11 ARQUIVO FitTrapArea.cpp

```
/*************************************************************

        FitTrapArea function implementation

                Check if radius is viable for current center and
        adjust it in case it is needed.

                Parameters:
                        -> none
                Return:
                        -> none

*************************************************************/

#include "Trap.hpp"

void UMATech::boarstrap::Trap::FitTrapArea(void){
        //Check if the trap fits inside the frame and adjust it
    if((_img.pos.center.x + _img.pos.radius) > _img.greyImage.size().width)
    {
        _img.pos.radius = _img.greyImage.size().width - _img.pos.center.x;
    }
    if((_img.pos.center.x - _img.pos.radius) < 0)
    {
        _img.pos.radius = _img.pos.center.x;
```

```
    }
    if((_img.pos.center.y + _img.pos.radius) > _img.greyImage.size().height)
    {
        _img.pos.radius = _img.greyImage.size().height - _img.pos.center.y;
    }
    if((_img.pos.center.y - _img.pos.radius) < 0)
    {
        _img.pos.radius = _img.pos.center.y;
        }
}
```

B.12 ARQUIVO GetCurrentFrame.cpp

```
/*************************************************************

       GetCurrentFrame function implementation

              Read a frame from device and turn it to grey scale.

              Parameters:
                     -> none
              Return:
                     -> none

*************************************************************/

#include "Trap.hpp"

void UMATech::boarstrap::Trap::GetCurrentFrame(void)
{
       //read image
    _video.read(_img.colourImage);

    //Color to Gray
    cv::cvtColor(_img.colourImage, _img.greyImage, CV_BGR2GRAY);
```

```
}
```

B.13 ARQUIVO GetMaxAreaIdx.cpp

```
/************************************************************

        GetMaxAreaIdx function implementation

                Find the moved region with the biggest area.

                Parameters:
                        -> none
                Return:
                        -> int : the id of the countour in countours
                                        vector

************************************************************/

#include "Trap.hpp"

int UMATech::boarstrap::Trap::GetMaxAreaIdx(void)
{
        //area of current biggest
    double candidateArea = 0.0f;
        //if has no biggest area yet
    if(-1 == _maxAreaIdx)
    {
                //iterate over all contours
        for(size_t i = 0; i < _roi.size(); i++)
        {
                        //if area is bigger than the current biggest
            if(_roi[i].size > candidateArea && _roi[i].inside)
            {
                                //new biggest area size and id
                candidateArea = _roi[i].size;
```

```
                _maxAreaIdx = i;
            }
        }
    }

        //retur id of the biggest area
    return _maxAreaIdx;
}
```

B.14 ARQUIVO Reset.cpp

```
/*************************************************************

        Reset function implementation

                Reset to default parameters.

                Parameters:
                        -> none
                Return:
                        -> none

*************************************************************/

#include "Trap.hpp"

void UMATech::boarstrap::Trap::Reset(void)
{
        //set default parameters
    _config.referenceSize = DEFAULT_SIZE;
    _img.pos.valid = false;
    _img.pos.radius = 150;
        //create a new background model
    SetBackground();
}
```

B.15 ARQUIVO RunImageProcessing.cpp

```
/************************************************************

       RunImageProcessing function implementation

              Treat image and create exclusion zones.

************************************************************/

#include "Trap.hpp"

/*
       Overloaded function

              Parameter:
                     -> input cv::Mat &MOG2Mask : mask for the motion
                                   regions
                     -> output cv::Mat &dest : matrix to store
                                   processed image
              Return:
                     -> none
*/
void UMATech::boarstrap::Trap::RunImageProcessing(cv::Mat &MOG2Mask, cv::Mat
    &dest)
{
    //At first the threshold value used to be the mean value
    //of gray pixels on the MOG2Mask, but after test and
    //carefully reviewing the documentation it was set to 127,
    //as the BackgroundSubtractor evaluate shadows to this value.
    //cv::Scalar meanValue = cv::mean(MOG2Mask);

    //For our purpose we'll use a binary threshold, meaning that
    //if the pixel value is greater than the mean, it will turn black
    //if it is lower, white
```

```
        //Threshold is not needed anymore, as the motion detection
           algorithm alredy takes care of it
        //cv::threshold(MOG2Mask, dest, 127, 255, 0);

    //The next step is apply erode to remove noises from our image
    //and reconstruct it using dilation.
    //In order to do that we first construct our erode and dilation
    //matrices
    cv::Mat erosionElement = cv::getStructuringElement(cv::MORPH_RECT, cv::Size
        (2*_config.erosionSize + 1, 2*_config.erosionSize + 1), cv::Point(_config.
        erosionSize, _config.erosionSize));
    cv::Mat dilationElement = cv::getStructuringElement(cv::MORPH_RECT, cv::Size
        (2*_config.dilationSize + 1, 2*_config.dilationSize + 1), cv::Point(_config.
        dilationSize, _config.dilationSize));

    //Then we apply it to the image
    cv::erode(MOG2Mask, dest, erosionElement);
    cv::dilate(MOG2Mask, dest, dilationElement);
}

/*
        Overloaded function

                Parameter:
                        -> none
                Return:
                        -> none
*/
void UMATech::boarstrap::Trap::RunImageProcessing(void)
{
    //call the other function for the internal region mask
        RunImageProcessing(_img.internalMotionMask, _img.dilatedImage);
    //call the other function for the external region mask
        RunImageProcessing(_img.externalMotionMask, _img.externalProc);
```

```
        //create exclusion zones
        SetExclusionZone();
}
```

B.16 ARQUIVO SendUserMssage.cpp

```
/************************************************************

        SendUserMessage function implementation

                Send a message to the user, warning about the
        deployment of the trap.

                Parameters:
                        -> none
                Return:
                        -> none

************************************************************/

#include "Trap.hpp"

void UMATech::boarstrap::Trap::SendUserMessage(void)
{
        //this is just a dummy function to exemplify how a message would
           be sent
        //to user if there was a transmission interface available

        //approximated number of individuals inside
        double nInside = _areaInside/(_config.refToTargetSize*_config.referenceSize);

        //message string to be sent
        std::string msg;
        msg += "1,"; //number saying the trap has been deployed
```

```
        msg += std::to_string(nInside); //number saying how many target there
            inside it
        msg += ",";

        //dummy transmitter class
        //TransmitterClass message();
        //message.write(msg);
}
```

B.17 ARQUIVO SetBackground.cpp

```
/************************************************************

        SetBackground function implementation

                Reset to default parameters.

                Parameters:
                        -> none
                Return:
                        -> none

************************************************************/

#include "Trap.hpp"

void UMATech::boarstrap::Trap::SetBackground(void)
{
        //get new frame
    GetCurrentFrame();
        //get trap area offset
    _poiOffset = TrapArea(_img);

        //create background model for internal region
        _internal.setBackground(_img.greyImage);
```

```
        //create background model for external region
        _external.setBackground(_img.greyImageExternal);
}
```

B.18 ARQUIVO SetCenterByImage.cpp

```
/************************************************************

        SetCenterByImage function implementation

                Set the trap center coordinates using image
        detection.

                Parameters:
                        -> none
                Return:
                        -> none

************************************************************/

#include "Trap.hpp"

void UMATech::boarstrap::Trap::SetCenterByImage(void)
{
    char input = 'n';

    //Set _useFullImage, to use the size of original image
    _useFullImage = true;
    //Set new background, as now we will use the full image
    SetBackground();

    std::cout << "Is the new center correct? (y/n/q)" << std::endl;

        //while the user doesn't quit or doesn't finish configuring
    while(input != 'y' && input != 'Y' && input != 'q' && input != 'Q')
```

```
    {
        if(input == 'n' || input == 'N')
        {
            //If center is not correct, reset background, allowing new
                positions
            SetBackground();
        }

                //call image processing functions
        ApplyBackground();
        RunImageProcessing();
        ExtractROIs();
        DrawROIs(true); //draw center as wells
        input = DisplayAndGetInput(0);
    }

        //if the user is happy with the center position
    if(input == 'y' || input == 'Y')
    {
                //calculate its position and validate it
        cv::Point tmp(_roi[GetMaxAreaIdx()].boundRect.x + _roi[GetMaxAreaIdx()].
            boundRect.width/2,
                _roi[GetMaxAreaIdx()].boundRect.y + _roi[GetMaxAreaIdx()].
                    boundRect.height/2);
        _img.pos.center = tmp;
        _img.pos.valid = true;
    }
    //Return to use only part of image
    _useFullImage = false;
    //Update the background
    SetBackground();

        //update exclusion zone position
        SetExclusionZone();
}
```

B.19 ARQUIVO SetCenterManually.cpp

```
/*************************************************************

       SetCenterManually function implementation

              Set the trap center coordinates manually.

              Parameters:
                     -> none
              Return:
                     -> none

*************************************************************/

#include "Trap.hpp"

void UMATech::boarstrap::Trap::SetCenterManually(void)
{
    char input = 'n';

        //store old configuration, in case the user wants to reset
           everything
        trap_area_t oldPos = _img.pos;

    //Set _useFullImage to use the size of original image
    _useFullImage = true;
    //Set new background as now we will use the full image
    SetBackground();

    std::cout << "Is the new center correct? (y/q)" << std::endl;

        //while the user doesn't quit or doesn't finish configuring
    while(input != 'y' && input != 'Y' && input != 'q' && input != 'Q')
    {
                if(input == 'a' || input == 'A'){
```

```
                //'a' move the center to the left
                _img.pos.center.x--;
                _img.pos.valid = true;
        }
        else if(input == 'd' || input == 'D'){
                //'d' move the center to the right
                _img.pos.center.x++;
                _img.pos.valid = true;
        }
        else if(input == 's' || input == 'S'){
                //'s' move the center down
                _img.pos.center.y++;
                _img.pos.valid = true;
        }
        else if(input == 'w' || input == 'W'){
                //'w' move the center up
                _img.pos.center.y--;
                _img.pos.valid = true;
        }

        //fit the trap area inside the image, in case it exceeds the
            borders
        FitTrapArea();
        //get new frame
        GetCurrentFrame();
        //draw a cross marking the center and draw the trap area
        cv::drawMarker(_img.colourImage, _img.pos.center, cv::Scalar(0, 0, 255)
            , cv::MARKER_CROSS, 10, 1);
        cv::circle(_img.colourImage, _img.pos.center, _img.pos.radius, cv::Scalar
            (0, 255, 0));
    input = DisplayAndGetInput(false);
}

if(input == 'q' || input == 'Q'){
```

```
            //return to old configuration, in case the user canceled the
                operation
            _img.pos = oldPos;
        }
        else{ //confirm new configuration otherwise
            FitTrapArea();
        std::cout << "New center set to: " << _img.pos.center.x << ", " <<
            _img.pos.center.y << std::endl;
        }

    //Return to use only part of image
    _useFullImage = false;
    //Update the background
    SetBackground();

        //update exclusion zone position
        SetExclusionZone();
}
```

B.20 ARQUIVO SetExclusionZone.cpp

```
/*************************************************************

        SetExclusionZone function implementation

            Set the exclusion zone masks.

            Parameters:
                -> none
            Return:
                -> none

*************************************************************/

#include "Trap.hpp"
```

```
void UMATech::boarstrap::Trap::SetExclusionZone(void){
        //create a zero filled matrix, the same size as the internal
            region of the trap
        _img.inExclusionZone = cv::Mat::zeros(_img.dilatedImage.size(), _img.
            dilatedImage.type());
        //draw a circle filled with white pixels, the same size as the
            trap
        cv::circle(_img.inExclusionZone,
                                cv::Point(_img.inExclusionZone.cols/2, _img.
                                    inExclusionZone.rows/2),
                                _config.runTimeConfig.radius,
                                cv::Scalar::all(255),
                                -1
                                );
        //draw another circle, but now filled with black pixels,
        //with radius smaller by 'frontierPercentage' percent
        cv::circle(_img.inExclusionZone,
                                cv::Point(_img.inExclusionZone.cols/2, _img.
                                    inExclusionZone.rows/2),
                                _config.runTimeConfig.radius*(1 - _config.
                                    frontierPercentage),
                                cv::Scalar::all(0),
                                -1
                                );
        //here we get a ring mask on the inner border of the trap
        //create a zero filled matrix, the same size as the whole image
        _img.outExclusionZone = cv::Mat::zeros(_img.externalProc.size(), _img.
            externalProc.type());
        //draw a circle filled with white pixels, with radius bigger by '
            frontierPercentage' percent
        cv::circle(_img.outExclusionZone,
                                _config.runTimeConfig.center,
                                _config.runTimeConfig.radius*(1 + _config.
                                    frontierPercentage),
```

```
                                cv::Scalar::all(255),
                                -1
                                );
        //draw another circle, but now filled with black pixels, the same
            size as the trap
        cv::circle(_img.outExclusionZone,
                                _config.runTimeConfig.center,
                                _config.runTimeConfig.radius,
                                cv::Scalar::all(0),
                                -1
                                );
        //here we get a ring mask on the outter border of the trap
        //Draw exclusion zone limits as red circles
        cv::circle(_img.colourImage, _img.pos.center, _config.runTimeConfig.radius*(1 -
            _config.frontierPercentage), cv::Scalar(0, 0, 255), 4);
        cv::circle(_img.colourImage, _img.pos.center, _config.runTimeConfig.radius*(1 +
            _config.frontierPercentage), cv::Scalar(0, 0, 255), 4);
}
```

B.21 ARQUIVO SetRadiusByImage.cpp

```
/*************************************************************

        SetRadiusByImage function implementation

                Set the trap radius using image detection.

                Parameters:
                        -> none
                Return:
                        -> none

*************************************************************/

#include "Trap.hpp"
```

```
void UMATech::boarstrap::Trap::SetRadiusByImage(void)
{
    char input = 'n';

    //Set _useFullImage to use the size of original image
    _useFullImage = true;
    //Set new background as now we will use the full image
    SetBackground();

    std::cout << "Is␣this␣the␣correct␣position?␣(y/n/q)" << std::endl;

        //while the user doesn't quit or doesn't finish configuring
    while(input != 'y' && input != 'Y' && input != 'q' && input != 'Q')
    {
        if(input == 'n' || input == 'N')
        {
            //If radius is not correct, reset background, allowing new
                positions
            SetBackground();
        }

                //call image processing functions
        ApplyBackground();
        RunImageProcessing();
        ExtractROIs();
        DrawROIs(true); //draw center as wells
        input = DisplayAndGetInput();
    }

        //if the user is happy with the radius
    if(input == 'y' || input == 'Y')
    {
                //calculate its size and validate it
```

```
        cv::Point tmp(_roi[GetMaxAreaIdx()].boundRect.x + _roi[GetMaxAreaIdx()].
            boundRect.width/2,
                _roi[GetMaxAreaIdx()].boundRect.y + _roi[GetMaxAreaIdx()].
                    boundRect.height/2);
        _img.pos.radius = (_img.pos.center.x - tmp.x) * (_img.pos.center.x - tmp.x);
        _img.pos.radius += (_img.pos.center.y - tmp.y) * (_img.pos.center.y - tmp.y);
        _img.pos.radius = std::sqrt(_img.pos.radius);
        _img.pos.valid = true;
        std::cout << "New radius set to: " << _img.pos.radius << std::endl;
    }
    //Return to use only part of image
    _useFullImage = false;
    //Update the background
    SetBackground();

        //update exclusion zone position
        SetExclusionZone();
}
```

B.22 ARQUIVO SetRadiusManually.cpp

```
/*************************************************************

        SetRadiusManually function implementation

                Set the trap radius manually.

                Parameters:
                        -> none
                Return:
                        -> none

*************************************************************/

#include "Trap.hpp"
```

```
void UMATech::boarstrap::Trap::SetRadiusManually(void)
{
    char input = 'n';

        //store old configuration, in case the user wants to reset
            everything
        trap_area_t oldPos = _img.pos;

    //Set _useFullImage to use the size of original image
    _useFullImage = true;
    //Set new background as now we will use the full image
    SetBackground();

    std::cout << "Is this the new radius correct? (y/q)" << std::endl;

        //while the user doesn't quit or doesn't finish configuring
    while(input != 'y' && input != 'Y' && input != 'q' && input != 'Q')
    {
        if(input == 'w' || input == 'W'){
                        //'w' increase the radius
            _img.pos.radius++;
                        _img.pos.valid = true;
        }
        else if(input == 's' || input == 'S'){
                        //'w' decrease the radius
            _img.pos.radius--;
                        _img.pos.valid = true;
        }

                //fit the trap area inside the image, in case it exceeds the
                    borders
                FitTrapArea();
                //get new frame
                GetCurrentFrame();
```

```
            //draw a cross marking the center and draw the trap area
            cv::drawMarker(_img.colourImage, _img.pos.center, cv::Scalar(0, 255, 0)
                , cv::MARKER_CROSS, 10, 1);
            cv::circle(_img.colourImage, _img.pos.center, _img.pos.radius, cv::Scalar
                (0, 0, 255));
        input = DisplayAndGetInput(false);
    }

    if(input == 'q' || input == 'Q'){
            //return to old configuration, in case the user canceled the
                operation
            _img.pos = oldPos;
    }
        else{ //confirm new configuration otherwise
            FitTrapArea();
        std::cout << "New radius set to: " << _img.pos.radius << std::endl;
        }

    //Return to use only part of image
    _useFullImage = false;
    //Update the background
    SetBackground();

        //update exclusion zone position
        SetExclusionZone();
}
```

B.23 ARQUIVO SetRefSizeByImage.cpp

```
/*************************************************************

        SetRefSizeByImage function implementation

            Set the minimum movement reference size using image
        detection.
```

```
            Parameters:
                    -> none
            Return:
                    -> none

*************************************************************/

#include "Trap.hpp"

void UMATech::boarstrap::Trap::SetRefSizeByImage(void)
{
    //The reference value is multiplied by a factor in
    //order to compensate for the size of shadows
        //the reference is the biggest movement on the image
    _config.referenceSize = _roi[GetMaxAreaIdx()].size * 0.90;
}
```

B.24 ARQUIVO SetRefSizeManually.cpp

```
/*************************************************************

       SetRefSizeManually function implementation

            Set the trap center coordinates manually.

            Parameters:
                    -> none
            Return:
                    -> none

*************************************************************/

#include "Trap.hpp"
```

```
void UMATech::boarstrap::Trap::SetRefSizeManually(void)
{
    char input = 'n';

        //store old configuration, in case the user wants to reset
            everything
        trap_config_t oldConfig = _config;

    //Set _useFullImage to use the size of original image
    _useFullImage = true;
    //Set new background as now we will use the full image
    SetBackground();

    std::cout << "Is this the reference size correct? (y/q)" << std::endl;

        //while the user doesn't quit or doesn't finish configuring
    while(input != 'y' && input != 'Y' && input != 'q' && input != 'Q')
    {
        if(input == 'w' || input == 'W'){
                        //'w' increase the ref size by 10%
            _config.referenceSize *= 1.1;
        }
        else if(input == 's' || input == 'S'){
                        //'s' decrease the ref size by 10%
            _config.referenceSize /= 1.1;
        }

                //get new frame
                GetCurrentFrame();
                //draw the reference size and configure it
                double side = cv::sqrt(_config.referenceSize);
                cv::Point pCenter(_img.colourImage.size().width/2, _img.colourImage.
                    size().height/2);
                cv::rectangle(_img.colourImage, pCenter - cv::Point(side/2, side/2),
                    pCenter + cv::Point(side/2, side/2), cv::Scalar(0, 0, 255));
```

```
        input = DisplayAndGetInput(false);
    }

    if(input == 'q' || input == 'Q'){
                //return to old configuration, in case the user canceled the
                    operation
                _config = oldConfig;
    }
        else{ //confirm new configuration otherwise
        std::cout << "New reference size set to: " << _config.referenceSize <<
            std::endl;
        }

    //Return to use only part of image
    _useFullImage = false;
    //Update the background
    SetBackground();
}
```

B.25 ARQUIVO TrapArea.cpp

```
/*************************************************************

        TrapArea function implementation

                Separate internal and external images.

                Parameters:
                        -> input trap_images_t &img : image matrices
                Return:
                        -> cv:Point : the center of the trap in
                                      relation to the whole image

*************************************************************/
```

```
#include "Trap.hpp"

cv::Point UMATech::boarstrap::Trap::TrapArea(trap_images_t &img)
{
        //default center coordinates
    cv::Point ret(0,0);
        //if the whole image is not to be used
    if(!_useFullImage)
    {
                //if configuration are not valid
        if(!img.pos.valid)
        {
                        //set center as the middle of the image
            cv::Point cen(img.greyImage.size().width/2, img.greyImage.size().height/2);
            img.pos.center = cen;
        }
        else
        {
            FitTrapArea();
        }
        //get the cv::Rect containing the circle:
        cv::Rect r(img.pos.center.x - img.pos.radius, img.pos.center.y - img.pos.radius
            , img.pos.radius*2, img.pos.radius*2);
        circle(img.colourImage, img.pos.center, img.pos.radius, cv::Scalar(255,0,0));

        //Obtain image ROI for both internal and external area of the trap
        cv::Mat roiInternal(img.greyImage, r);
        cv::Mat roiExternal(img.greyImage);

        // make a black mask, for the internal roi:
        cv::Mat maskInternal(roiInternal.size(), roiInternal.type(), cv::Scalar::all(0));
        //and a white mask for the external
        cv::Mat maskExternal(roiExternal.size(), roiExternal.type(), cv::Scalar::all(255)
            );
        //with a white, filled circle in it for the internal roi:
```

```
        circle(maskInternal, cv::Point(img.pos.radius,img.pos.radius), img.pos.radius, cv
            ::Scalar::all(255), −1);
        //and a black circle for the external
        circle(maskExternal, img.pos.center, img.pos.radius, cv::Scalar::all(0), −1);

        // combine roi & mask:
        img.greyImage = roiInternal & maskInternal;
        img.greyImageExternal = roiExternal & maskExternal;

        //Return curent POI center
        ret = img.pos.center − cv::Point(img.pos.radius, img.pos.radius);
    }
    else
    {
        //Do nothing, return default center
    }

        //return the center coordinates
    return ret;
}
```

APÊNDICE C – ARQUIVO DE CONFIGURAÇÃO

Neste apêndice está apresentado um exemplo de arquivo de configuração, que pode ser modificado pelo usuário e passado para o *software* como parâmetro no momento de execução.

C.1 ARQUIVO config.conf

```
//Arquivo de configuracao do usuario, altere apenas a linha 3 respeitando
    a seguinte formatacao:
//caminho_dev,matriz_erosion,matriz_dilation,tamanho_alvo,minimo_dentro,
   maximo_fora,distancia_da_borda,center_x,center_y,radius,refSize,
   detectThreshold,detectSimilars,delayToDeploy,caminho_sysconfig,
/dev/video0,3,5,3,5,1,10,382,240,150,200,20,3,2,,
```

APÊNDICE D – CÓDIGO FONTE DA BIBLIOTECA DE GPIO

Neste apêndice estão apresentados os códigos e os arquivos de *README* e licença de uso da biblioteca de GPIO utilizada neste trabalho, que foi desenvolvida por terceiros.

D.1 ARQUIVO GPIOClass.h

```
#ifndef GPIO_CLASS_H
#define GPIO_CLASS_H

#include <string>
#include <iostream>
#include <sstream>
#include <unistd.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

using namespace std;
/* GPIO Class
 * Purpose: Each object instatiated from this class will control a GPIO
    pin
 * The GPIO pin number must be passed to the overloaded class constructor
 */
class GPIOClass
{
public:
```

```
        GPIOClass();
        GPIOClass(string gnum);
        ~GPIOClass();
    int setdir_gpio(string dir);
    int setval_gpio(string val);
    int getval_gpio(string& val);
    string get_gpionum();
private:
    int export_gpio();
        int unexport_gpio();

        int valuefd;
        int directionfd;
        int exportfd;
        int unexportfd;
        string gpionum;

};

#endif
```

D.2 ARQUIVO GPIOClass.cpp

```
#include "GPIOClass.h"

using namespace std;

GPIOClass::GPIOClass():valuefd(-1),directionfd(-1),exportfd(-1),unexportfd(-1),
    gpionum("4")
{
        //GPIO4 is default
        this->unexport_gpio();
        this->export_gpio();
}
```

```
GPIOClass::GPIOClass(string gnum):valuefd(-1),directionfd(-1),exportfd(-1),
    unexportfd(-1),gpionum(gnum)
{
        //Instatiate GPIOClass object for GPIO pin number "gnum"
        this->unexport_gpio();
        this->export_gpio();
}

GPIOClass::~GPIOClass()
{
        this->unexport_gpio();
}


int GPIOClass::export_gpio()
{
        int statusVal = -1;
        string exportStr = "/sys/class/gpio/export";
        this->exportfd = statusVal = open(exportStr.c_str(), O_WRONLY|O_SYNC)
            ;
        if (statusVal < 0){
                perror("could not open SYSFS GPIO export device");
        //exit(1);
        }

        stringstream ss;
        ss << this->gpionum;
        string numStr = ss.str();



        statusVal = write(this->exportfd, numStr.c_str(), numStr.length());
        if (statusVal < 0){
                std::cout << numStr << std::endl;
                perror("could not write to SYSFS GPIO export device");
```

```
        //exit(1);
        }

        statusVal = close(this->exportfd);
        if (statusVal < 0){
                perror("could not close SYSFS GPIO export device");
        //exit(1);
        }

    return statusVal;
}

int GPIOClass::unexport_gpio()
{
        int statusVal = -1;
        string unexportStr = "/sys/class/gpio/unexport";
        this->unexportfd = statusVal = open(unexportStr.c_str(), O_WRONLY|
            O_SYNC);
        if (statusVal < 0){
                perror("could not open SYSFS GPIO unexport device");
        }

        stringstream ss;
        ss << this->gpionum;
        string numStr = ss.str();
        statusVal = write(this->unexportfd, numStr.c_str(), numStr.length());
        if (statusVal < 0){
                perror("could not write to SYSFS GPIO unexport device");
        }

        statusVal = close(this->unexportfd);
        if (statusVal < 0){
                perror("could not close SYSFS GPIO unexport device");
        }
```

```
        return statusVal;
}

int GPIOClass::setdir_gpio(string dir)
{
        int statusVal = -1;
        string setdirStr ="/sys/class/gpio/gpio" + this->gpionum + "/
            direction";
        std::cout << "/sys/class/gpio/gpio" + this->gpionum + "/direction"
            << std::endl;

        this->directionfd = statusVal = open(setdirStr.c_str(), O_WRONLY|O_SYNC
            ); // open direction file for gpio
                if (statusVal < 0){
                        perror("could␣not␣open␣SYSFS␣GPIO␣direction␣device");
                }

        if (dir.compare("in") != 0 && dir.compare("out") != 0 ) {
                fprintf(stderr, "Invalid␣direction␣value.␣Should␣be␣\"in\"␣or␣\"
                    out\".␣\n");
        }

        statusVal = write(this->directionfd, dir.c_str(), dir.length());
        if (statusVal < 0){
                perror("could␣not␣write␣to␣SYSFS␣GPIO␣direction␣device");
        }

        statusVal = close(this->directionfd);
        if (statusVal < 0){
                perror("could␣not␣close␣SYSFS␣GPIO␣direction␣device");
        }

            return statusVal;
}
```

```
int GPIOClass::setval_gpio(string val)
{

    int statusVal = -1;

        std::cout << "/sys/class/gpio/gpio" + this->gpionum + "/value" <<
            std::endl;

        string setValStr = "/sys/class/gpio/gpio" + this->gpionum + "/value";

        this->valuefd = statusVal = open(setValStr.c_str(), O_WRONLY|O_SYNC);
        if (statusVal < 0){
                perror("could not open SYSFS GPIO value device");
        }

        if (val.compare("1") != 0 && val.compare("0") != 0 ) {
                fprintf(stderr, "Invalid  value. Should be \"1\" or \"0\". \n");
                //exit(1);
        }

        statusVal = write(this->valuefd, val.c_str(), val.length());
        if (statusVal < 0){
                perror("could not write to SYSFS GPIO value device");
        }

        statusVal = close(this->valuefd);
        if (statusVal < 0){
                perror("could not close SYSFS GPIO value device");
        }

            return statusVal;
}
```

```
int GPIOClass::getval_gpio(string& val){

        string getValStr = "/sys/class/gpio/gpio" + this->gpionum + "/value";
        char buff[10];
        int statusVal = -1;
        this->valuefd = statusVal = open(getValStr.c_str(), O_RDONLY|O_SYNC);
        if (statusVal < 0){
                        perror("could not open SYSFS GPIO value device");
        }

        statusVal = read(this->valuefd, &buff, 1);
        if (statusVal < 0){
                perror("could not read SYSFS GPIO value device");
        }

        buff[1]='\0';

        val = string(buff);

        if (val.compare("1") != 0 && val.compare("0") != 0 ) {
                fprintf(stderr, "Invalid  value read. Should be \"1\" or \"0\". \
                    \n");
        }

        statusVal = close(this->valuefd);
        if (statusVal < 0){
                perror("could not close SYSFS GPIO value device");
        }

        return statusVal;
}


string GPIOClass::get_gpionum(){
```

```
return this->gpionum;

}
```

D.3 ARQUIVO LICENSE

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights.

These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed

under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in

whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine−readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine−readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as

distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you

may not distribute the Program at all. For example, if a patent license would not permit royalty−free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will

be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE

PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

Copyright (C) 2013 halherta

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License along
with this program; if not, write to the Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA 02110−1301 USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show
w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate
parts of the General Public License. Of course, the commands you use may
be called something other than ‘show w’ and ‘show c’; they could even be
mouse−clicks or menu items−−whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a ”copyright disclaimer” for the program, if

necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program
'Gnomovision' (which makes passes at compilers) written by James Hacker.

{signature of Ty Coon}, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License.

D.4 ARQUIVO README.md

```
RaspberryPi-GPIOClass-v2
==================================

Author: Hussam Al-Hertani

Description: The GPIOClass C++ class enables the Raspberry Pi its onboard GPIOs
    via the sysfs interface.

================================================

For more information please visit: http://hertaville.com/2012/11/18/introduction-to-
    accessing-the-raspberry-pis-gpio-in-c/

================================================

To build example natively on the Raspberry Pi:

g++ -fpermissive -Wall GPIOClass.cpp GPIOtest.cpp -o outBin
```

Make sure that the GPIOClass.h header file is in the work (RaspberryPi−GPIOClass−v2) directory.

APÊNDICE E – GUIA DE USO

A fim de auxiliar a instalação e testes com o protótipo, este apêndice propõe-se a explicar quais são os ajustes possíveis e como realizar esses ajustes durante a execução do programa e instalação da armadilha. Entretanto esse guia não deve ser entendido como um manual destinado ao usuário final, mas sim uma material de apoio a desenvolvedores que desejam dar continuidade ao projeto.

Este guia assume conhecimentos de Linux, acesso remoto via SSH e conhecimentos básicos de informática. Assume-se ainda que o código descrito neste projeto e mais especificamente nos apêndices A, B, C e D já foi compilado e o arquivo executável resultante foi transferido para um Rasberry Pi. Após isto, estando conectado ao Raspberry Pi via uma seção SSH com suporte a *X11 forwarding*, deve-se executar a aplicação *boarstrap* passando como pârametro o arquivo de configuração introduzido na seção 4.1.1.5. Seráo exibidas 3 janelas com os seguintes nomes:

- **"Original"**: Mostrada na Figura 17. Nesta janela é possível visualizar o campo de visão da câmera onde o círculo em azul representa os limites da armadilha, os círculos vermelhos delimitam as zonas de exclusão e o contorno vermelho ou verde indica uma área movimentada.

- **"Frame"**: Apresentada na Figura 18. Esta janela mostra o *frame* resultante da etapa de tratamento da imagem para área interna da armadilha. Em branco é possivel visualizar a região movimentada. Nota-se também que movimento é detectado apenas no interior do círculo azul.

- **"External Frame"**: Figura 19. Semelhante a tela *"Frame"*, esta exibe o resultado do tratemento para a região externa a armadilha, com o movimento detectado da área externa do círculo azul apenas.

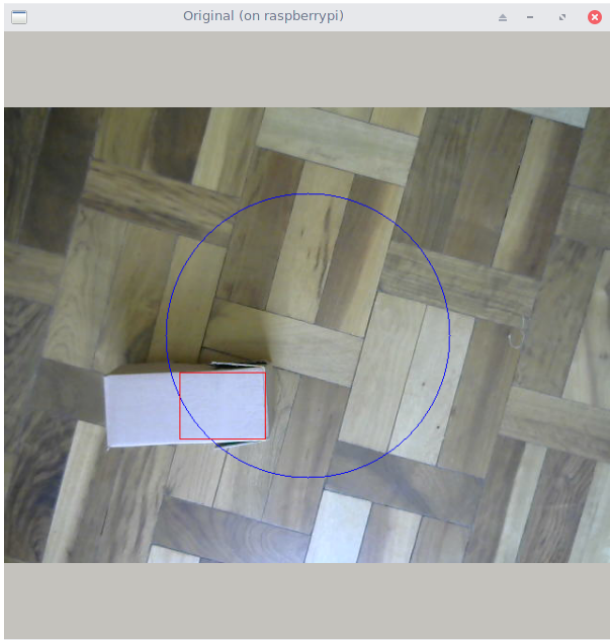

Figura 17: Tela “Original”

Fonte: Autoria própria.

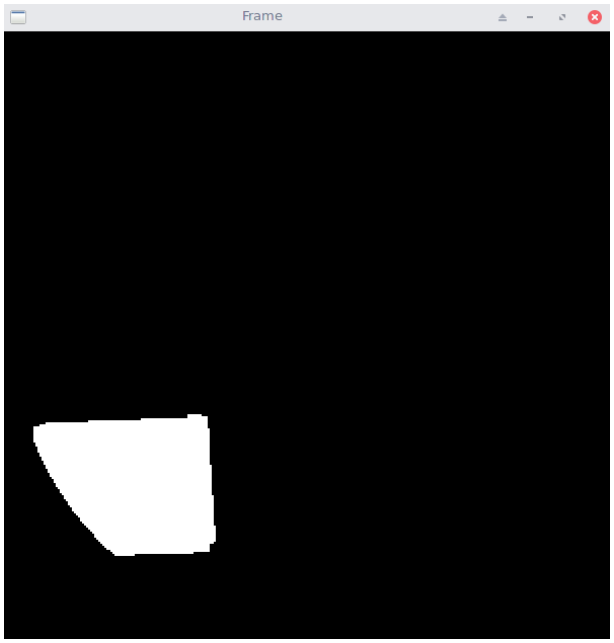

Figura 18: Tela “Frame”

Fonte: Autoria própria.

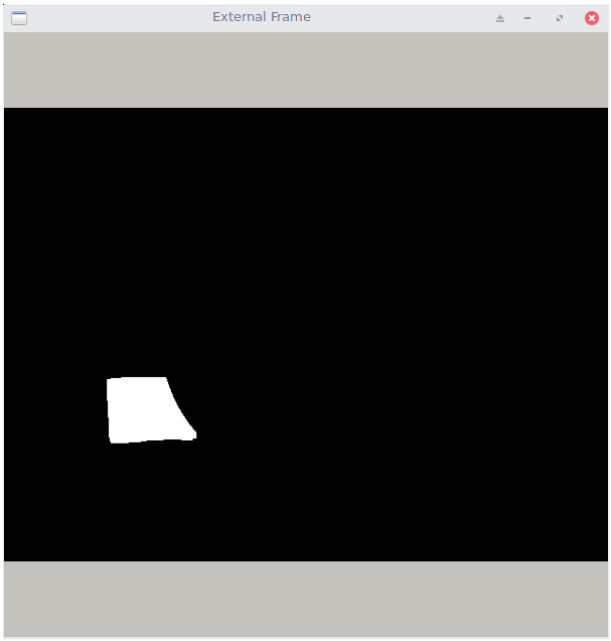

Figura 19: Tela "*External Frame*"
Fonte: Autoria própria.

E.1 CONFIGURAÇÃO DO TAMANHO DE REFERÊNCIA

A primeira configuração que pode ser alterada é o tamanho de referência. Há duas formas de modificar este parâmetro, um é através de reconhecimento de imagem e ou é manualmente. Para proceder da primeira forma, deve-se posicionar um objeto a ser usado como referência dentro do campo de visão da câmera, preferencialmente na região central, então pressionar '2'. O novo referencial de tamanho será indicado pelo retângulo vermelho, Figura 20, que representa o maior movimento na tela. Caso existam outros objetos esses estarão indicados por um retângulo verde e não seram utilizados como referencial.

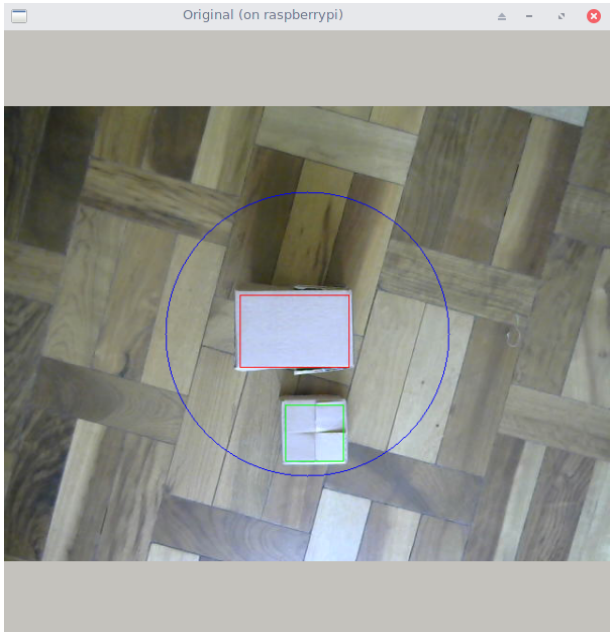

Figura 20: Processo de escolha de novo referencial de tamanho.
Fonte: Autoria própria.

A segunda forma de ajustar a referência de tamanho é, pressionar a tecla '5'. Aparecerá na janela de image, então, um quadrado vermelho representando o tamanho de referência atual. O tamanho deste quadrado pode ser alterado pressionando-se as teclas 'w' e 's'. Note-que a área escolhida é igual à área total do quadrado vermelho. A mensagem "Is this the reference size correct? (y/q)"será exibida no terminal durante a execução deste ajuste, para confirmar o tamanho desejado deve-se pressionar 'y', enquanto que para descartar as alterações e retornar aos valores anteriores, deve-se pressionar 'q'.

A implementação destas funcionalidades pode ser encontrada no apêndice B, nas funções **SetRefSizerByImage** e **SetRefSizerByManually**.

E.2 CONFIGURAÇÃO DA POSIÇÃO DO CENTRO DA ARMADILHA

Também é possivel reposicionar o centro da armadilha, por imagem e manualmente. Para reposicionar por imagem, deve-se pressionar '3'. Neste modo, as linhas que dividem a imagem em áreas de interesse são desativadas, permitindo que o novo centro seja posicionado em qualquer ponto da imagem. Deve-se ter em mente que caso o novo centro torne inviável para o tamanho atual de raio, este será modificado para o maior valor possível dentro do campo de visão da câmera. Assim como para o tamanho de referência, o centro é determinado pela posição do objeto de maior tamanho que se encontre na image. Durante o reposicionamento o terminal exibirá a mensagem "Is the new cen-

ter correct? (y/n/q)"e o ponto que será configurado como centro é exibido em vermelho dentro de um retângulo, também vermelho, como mostrado na Figura 21.

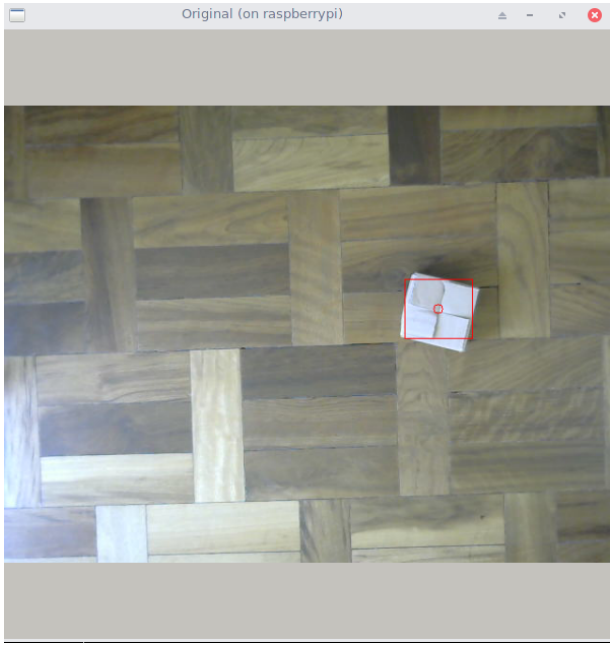

Figura 21: Processo de escolha de novo centro.
Fonte: Autoria própria.

Se o novo centro foi identificado com sucesso o usuário deve pressionar 'y' no terminal. Se desejar reiniciar o *background* e apresentar um novo centro pressiona-se 'n'. E, para cancelar, entrar a opção 'q'. A Figura 22 mostra o resultado da nova configuração escolhida.

Figura 22: Centro da armadilha reposicionado.
Fonte: Autoria própria.

Os mesmos resultados podem ser obtidos através da configuração manual, pressionando-se a tecla ‘6’ na janela principal do programa. O centro aparecerá como um ‘+’ dentro de um cículo azul, que pode ser movimentado através das teclas ‘a’, ‘w’, ‘s’ e ‘d’, sendo que o raio se auto-ajusta caso o cícurlo não caiba na imagem. A mesma mensagem de confirmação do parâmetro anterior será apresentada, para aceitar as modificações deve-se pressionar ‘y’ e para rejeitar ‘q’.

E.3 CONFIGURAÇÃO DO TAMANHO DO RAIO DA ARMADILHA

O último parâmetro configurável em tempo real é o raio de ação da armadilha, podendo ser ajustado das mesmas formas que os anteriores. Para ajustar o raio por imagem, pressione ‘4’. A menssagem “Is this the correct position? (y/n/q)”será exibida no terminal. Em seguida localize um objeto de referência na posição desejada, assim como para configurar o centro. A posição será indicada por um ponto vermelho dentro de um retângulo também vermelho: Figura 23.

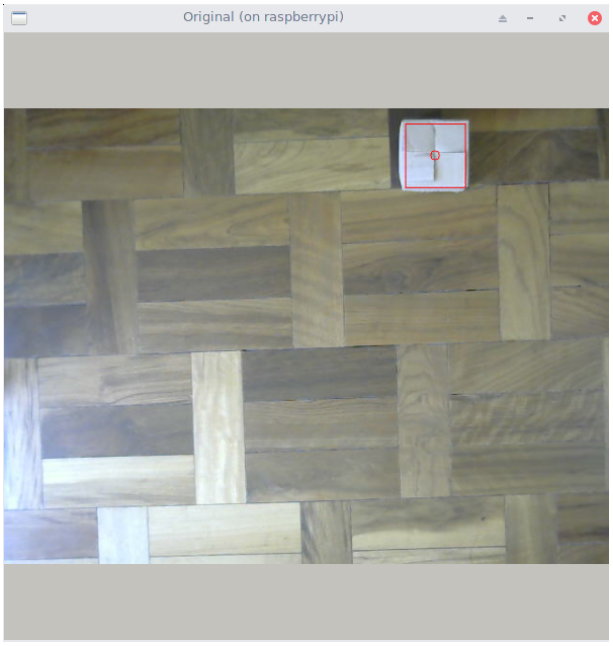

Figura 23: Processo de ajuste do raio.
Fonte: Autoria própria.

Se o novo raio foi ajustado com sucesso o usuário deve pressionar 'y' no terminal. Se desejar reiniciar o *background* e apresentar um novo ajuste pressiona-se 'n'. E para cancelar, entrar a opção 'q'. A Figura 24 mostra o novo raio escolhido.

Figura 24: Resultado final com novo centro e raio.
Fonte: Autoria própria.

Para a alteração manual do raio, deve-se pressionar '7'. Será exebido um círculo

indicando o tamanho da área da armadilha que está configurada atualmente, com centro e raio corretos. O tamanho deste círculo, ou seja, seu raio, pode ser alterado através das teclas ‘w’ e ‘s’, sendo que o raio pára de aumentar caso o cícurlo não caiba na imagem. A mesma mensagem de confirmação é exibida. Para aceitar as modificações deve-se pressionar ‘y’ e para rejeitar ‘q’.

Embora não seja obrigatório, recomenda-se configurar o centro e raio da armadilha sempre que uma nova instalação ou manutenção seja realizada na armadilha.

E.4 CONFIGURAÇÕES ADICIONAIS

Permite-se também que, pressionando ‘0’, as configurações de fábrica sejam restauradas, conforme mostrado na Figura 25. E que, pressionando-se ‘1’, o modelo de *background* seja atualizado.

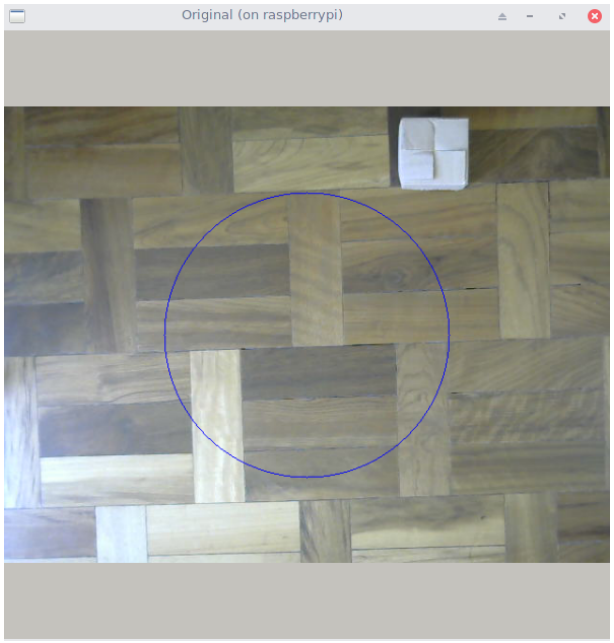

Figura 25: Configurações de fábrica restauradas.
Fonte: Autoria própria.